PROYECTO FIN DE CARRERA

DESARROLLO DE UN SISTEMA DE TRANSMISIÓN DE VÍDEO A TRAVÉS DE DDS

Universidad Carlos III de Madrid
Escuela Politécnica Superior

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN: IMAGEN Y SONIDO

AUTOR: Juan Carlos Moreno Rodríguez
TUTORA: Dra. Iria Manuela Estévez Ayres

Título: Desarrollo de un sistema de transmisión de vídeo a través de DDS

Autor: Juan Carlos Moreno Rodríguez

Tutora: Dra. Iría Manuela Estévez Ayres

La defensa del presente Proyecto Fin de Carrera se realizó el día 18 de Octubre de 2011; siendo calificada por el siguiente tribunal:

Presidente: *Jesús Arias Fisteus*

Secretario: *Antonio de la Oliva*

Vocal: *Jose Miguel Leiva*

Habiendo obtenido la siguiente calificación:

Calificación:

Presidente *Secretario* *Vocal*

Dedicado a todos los que

me han ayudado y confiado en mí,

a mi familia y en especial a Lorena, gracias por estar ahí!

RESUMEN

La transmisión de datos multimedia, a día de hoy, es cada vez más utilizada y necesaria en entornos particulares y empresariales. La distribución de este tipo de datos supone un avance en la comunicación entre usuarios produciendo una mejora en el intercambio de información.

Por ello, en el presente proyecto, se quiere realizar una aproximación al intercambio de datos multimedia en sistemas distribuidos y su reproducción en tiempo real. Para ello, se ha creado una aplicación que, mediante el paradigma de publicación/suscripción, realiza la transmisión de los datos de vídeo desde un nodo para la posterior obtención y reproducción de los mismos en otro nodo que se encuentra en su misma red.

Para la realización de este proyecto, se ha utilizado una versión libre de un sistema de distribución de datos como es OpenSplice y el uso de herramientas para la descodificación y visualización de los datos como son FFmpeg y SDL.

ABSTRACT

The transmission of multimedia data, to date, is increasingly used and necessary in private and business environments. Distribution of this data type represents a breakthrough in communication between users leading to an improvement in the exchange of information.

Thus, in the present project, we want to create an approach to multimedia data exchange in distributed systems and real time playback. For this purpose, an application has been created that, by means of the paradigm of publish/ subscribe, makes the transmission of video data from one node to the subsequent acquisition and reproduction of the data in another node in its network.

For this project, we used a free version of a data distribution system OpenSplice and tools for the decoding and display of data such as FFmpeg and SDL.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE GRÁFICAS

LISTA DE ACRÓNIMOS

- **DDS** *Data Distribution System (Sistema Distribuido de Datos)*
- **IDL** *Interface description language (Descripción de Lenguaje de Interfaz)*
- **API** *Application Programming Interface (Interfaz de Aplicación de Programa)*
- **NOS** *Network Operating System (Sistema Operativo de Red)*
- **MOM** *Message-oriented middleware (Middleware Orientado a Mensajes)*
- **DAM** *Data Access Middleware (Middleware de Acceso de Datos)*
- **GPS** *Global Positioning System (Sistema de Posicionamiento Global)*
- **DBMS** *Data Base Management System (Sistema de Gestión de Bases de Datos)*
- **LIS** *Laboratory Information System (Sistema Informático del Laboratorio)*
- **QoS** *Quality of Service (Calidad de Servicio)*
- **DP** *Domino Público*
- **P** *Publisher (Publicador)*
- **S** *Subscriber (Subscriptor)*
- **DW** *Data Writer (Escritor de Datos)*
- **DR** *Data Reader (Lector de Datos)*
- **T** *Topic (Tópico)*
- **UML** *Unified Modeling Language (Modelado Unificado de Lenguaje)*
- **RTSP** *Real-Time Streaming Protocol (Protocolo de Envío en Tiempo Real)*
- **RTP** *Real-Time Transport Protocol (Protocolo de Transporte en Tiempo Real)*
- **UDP** *User Datagram Protocol (Protocolo de Datagramas de Usuario)*
- **IP** *Internet Protocol (Protocolo de Internet)*
- **MPEG-2** *Moving Pictures Experts Group 2 (Grupo de Expertos en Imágenes en Movimiento 2)*
- **ISO** *International Organization for Standardization (Organización Internacional de Estandarización)*
- **TDT** *Televisión Digital Terrestre*
- **SVCD** *Super Video Compact Disc (Disco Compacto de Súper Video)*

- **DVD** *Digital Video Disc (Disco de Vídeo Digital)*
- **HDTV** *High Definition Television (Televisión de Alta Definición)*
- **AAC** *Advanced Audio Coding (Codificación de Audio Avanzada)*
- **TS** *Transport Stream (Flujo de Transporte)*
- **PS** *Program Stream (Flujo de Programa)*
- **PES** *Packetized Elementary Stream (Flujo Elemental de Paquete)*
- **PID** *Packet Identifier (Identificador de Paquete)*
- **PCR** *Program Clock Reference (Referencia del Reloj del Programa)*
- **MHz** *Megaherzio*
- **KHz** *Kiloherzio*
- **SQL** *Structured Query Language (Lenguaje Estructurado de Consulta)*
- **DCPS** *Data-Centric Publish-Subscribe (Centro de Datos Publicador-Subscriptor)*
- **GNU** *General Public License (Licencia Pública General)*
- **HTTP** *Hypertext Transfer Protocol (Protocolo de Transporte de Hipertexto)*
- **SDL** *Simple DirectMedia Layer (Capa Simple Multimedia Directa)*
- **PPM** *Portable Pixel Map (Mapa Portable de Pixel)*
- **RGB** *Red Green Blue (Rojo Verde Azul)*
- **PC** *Personal Computer (Ordenador Personal)*

Capítulo 1: Introducción y Motivación

"Los grandes trabajos no son hechos por la fuerza, sino por la perseverancia".

Samuel Johnson *(1709-1784) Poeta inglés*

En este capítulo se realiza una explicación breve sobre el actual modelo de transmisión de vídeo, las motivaciones que han llevado a realizar este proyecto y la estructura en la que se ha organizado el mismo.

1.1 Introducción

Durante mucho tiempo el vídeo ha sido un importante medio de comunicación y entretenimiento. Desde sus comienzos se utilizó la tecnología analógica para su captura, transmisión, almacenamiento y reproducción. El ejemplo más representativo es la televisión convencional, siendo ésta el principal medio de comunicación en la mayoría de los países. Sin embargo, con la llegada de la tecnología digital y la masificación de los computadores, la transmisión digital de contenido multimedia es su sucesor natural.

Últimamente, el *streaming*, se ha ido imponiendo como forma de visualización de contenidos, tecnología que permite la transmisión de información multimedia en tiempo real con un control de sesión dinámico. Con esta alternativa, no se usa el máximo ancho de banda disponible por el cliente para descargar y visualizar el medio, sino que tan sólo se usa el ancho de banda necesario para ir reproduciendo el medio en tiempo real. Además, no se produce una descarga completa del medio, sino que conforme se descarga se va descartando una vez ha sido utilizado para la reproducción.

Por otro lado, los sistemas distribuidos tienen una gran importancia en la sociedad actual debido, fundamentalmente, a la posibilidad que ofrecen para comunicar remotamente a usuarios y a éstos con los servicios a los que tradicionalmente acudían de forma presencial. El auge de Internet hace posible el aumento de la capacidad de cómputo de los servicios que se ofrecen en la red que son utilizados por multitud de personas y cuya carga se distribuye, en muchas ocasiones, entre muchas máquinas.

Ante dicho entorno, donde es evidente que existen multitud de nodos heterogéneos, es necesario utilizar técnicas para saltar los obstáculos que a priori plantea la comunicación entre distintas plataformas *hardware* y *software*. El *middleware* tiene un papel fundamental ante este panorama ya que es la capa de software que permite esconder la heterogeneidad de las plataformas que comunican.

En este marco surge la tecnología *DDS (Data Distribution System)*, pensada para poder conectar máquinas remotas con requisitos de tiempo real, tiempos deterministas, tolerancia a fallos y que permite trabajar con plataformas heterogéneas. Esta tecnología es novedosa y compleja. No es única en su campo, pero por lo demostrado hasta el momento si es una de las más eficientes. Su estudio, análisis, desarrollo y aplicación en nuevos sistemas está orientado al desarrollo tecnológico más puntero de la comunicación y de la computación: los sistemas de tiempo real distribuidos y desacoplados.

1.2 Motivación

El principal objetivo de este proyecto es el estudio del funcionamiento de la tecnología de *middleware* en tiempo real DDS en su implementación proporcionada por la empresa PrismTech. El estudio quiere centrarse en cómo se desarrollan aplicaciones para esta tecnología explicando los pasos que hay que seguir y los conocimientos que son necesarios para el desarrollo y la problemática que genera la creación de aplicaciones.

Por otro lado, se quiere realizar un acercamiento al uso del *streaming* como método de transmisión de vídeo. Para ello se realiza un estudio de las herramientas que ayudan a la reproducción de los contenidos audiovisuales como son FFMpeg en el caso de la codificación y el tipo de contenedores, y SDL en la visualización de contenidos multimedia.

El desarrollo de todo esto conlleva una labor de estudio para tener un amplio conocimiento de las tecnologías que existen en la actualidad en el mercado para poder decantarse por una solución final que pueda realizarse en lenguajes de amplia distribución como son Java, C o C++. En concreto, en este proyecto, la práctica totalidad del código se ha desarrollado en C, apoyándose en otro lenguajes, tales como *IDL (Interface Description Language)* y proporcionar así interoperabilidad a la aplicación.

En una última mejora, también se propuso el estudio y desarrollo de la aplicación para una posible incorporación a un terminal móvil, en este caso Maemo. Finalmente, tuvo que descartarse esta opción debido a la incompatibilidad de instalación de la tecnología DDS en el terminal.

En definitiva, aunque el proyecto culmine con la realización de una aplicación final, el objetivo no es tanto el desarrollo de esa aplicación sino abrir el camino a la realización de otras similares ofreciendo la experiencia y conocimientos desarrollados a lo largo del proyecto de una tecnología con gran capacidad de integración y un poderoso *API (Aplication Programing Interface)* que facilita en gran medida la creación de programas.

1.3 Estructura del proyecto

La presente memoria se estructura en varios bloques. En primer lugar, en el *Capítulo 2*, un estado del arte que describe cómo se encuentran actualmente los sistemas de distribución, modelos y características, así como una introducción al concepto de *middleware* como capa intermedia. También se realiza un estudio sobre sistemas multimedia y el *streaming* como forma de transmisión del mismo, ahondando en los diferentes formatos contenedores y códecs, para finalizar con Mpeg, el formato más utilizado en transmisión de vídeo. En definitiva, se describe el contexto técnico en el que se insertan las aplicaciones que componen este proyecto.

A continuación, en el *Capítulo 3*, se incluye una descripción de las herramientas y tecnologías empleadas para el desarrollo de la aplicación. La plataforma de distribución sobre la que se trabaja y las primeras aplicaciones realizadas con ella para la toma de contacto con el entorno, así como las herramientas para la manipulación y visualización de datos multimedia. Esta parte comprende el proceso de adquisición del denominado *know- how* (o saber hacer), es decir, comenzar un proyecto sin conocer nada de la plataforma, las herramientas a utilizar, ni del entorno de desarrollo y poder, finalmente, terminar con conocimientos de todo lo mencionado.

El *Capítulo 4* es el bloque principal del proyecto donde, se describe el desarrollo de la aplicación final. En este apartado se describe el entorno en el que se ha desarrollado, la arquitectura y las decisiones de diseño que se ha tomado para su realización y una descripción de los aspectos especialmente interesantes del desarrollo con las partes más relevantes del código.

Tras el desarrollo de la aplicación, en el *Capítulo 5*, se incorporan una serie de pruebas a las que se ha sometido la aplicación para comprobar su correcto funcionamiento y rendimiento.

Las dos últimas secciones están reservadas para las conclusiones a las que se ha llegado después de la realización del proyecto y las posibles líneas futuras de trabajo en las que se puede continuar, ambos, se muestran en el *Capítulo 6*, además, el presupuesto y el tiempo que se ha dedicado de forma global al proyecto se encuentra en el *Capítulo 7*.

En la sección de *Apéndices* se realizan las explicaciones pertinentes para la instalación de los diferentes elementos que intervienen en el desarrollo y así poder montar el entorno de forma correcta. Además, se comenta el intento de incorporación de la aplicación en un terminal Maemo.

Capítulo 2: Estado del Arte

"Nunca consideres el estudio como una obligación, sino como una oportunidad para penetrar en el bello y maravilloso mundo del saber".
Albert Einstein *(1879-1955) Científico alemán*

En este capítulo se hablará sobre las diferentes tecnologías que se usan en este Proyecto de Fin de Carrera y que son necesarias para su realización. Se puede diferenciar claramente tres partes. En la primera parte, se estudia el Middleware como elemento de conexión de aplicaciones y los diferentes tipos que existen. En la segunda, se pretende comprender qué es un sistema distribuido, sus características y funcionamiento. En la tercera y última parte se realiza un acercamiento a la tecnología de streaming como sistema de transmisión de vídeo, conociendo los diferentes codificadores y contenedores de contenido que pueden ser utilizados.

2.1 Middleware

El término *Middleware* se refiere a un software que da la capacidad a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos. Éste simplifica el trabajo de los programadores en la tarea de generar las conexiones que son necesarias en los sistemas distribuidos. De esta forma se provee una solución que mejora la calidad de servicio, seguridad, envío de mensajes, etc, dentro de las aplicaciones en las que actúan *[1][2]*.

Así, el *middleware* se comporta como una capa de abstracción de software distribuida, que se sitúa entre la capa de aplicación y las capas inferiores (sistema operativo y red). El *middleware* abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una *API* para la fácil programación y manejo de aplicaciones distribuidas. Dependiendo del problema a resolver y de las funcionalidades necesarias, serán útiles diferentes tipo de servicios de *middleware*.

El origen de la palabra *middleware* se remonta al año 1968, en donde la palabra es usada durante la 1968 *NATO Software Engineering Conference*, *[3]* siendo una idea de cómo conectar el nuevo software con sistemas más antiguos. Durante las décadas previas a los 1990s, fue solamente descrito como un software para la gestión de conexión en redes. Para cuando las tecnologías en redes alcanzaron una penetración y visibilidad suficiente el software *middleware* había evolucionado en un conjunto de paradigmas y servicios. De esta forma se estaba ofreciendo una manera más fácil, robusta y controlable para construir aplicaciones distribuidas *[4]*.

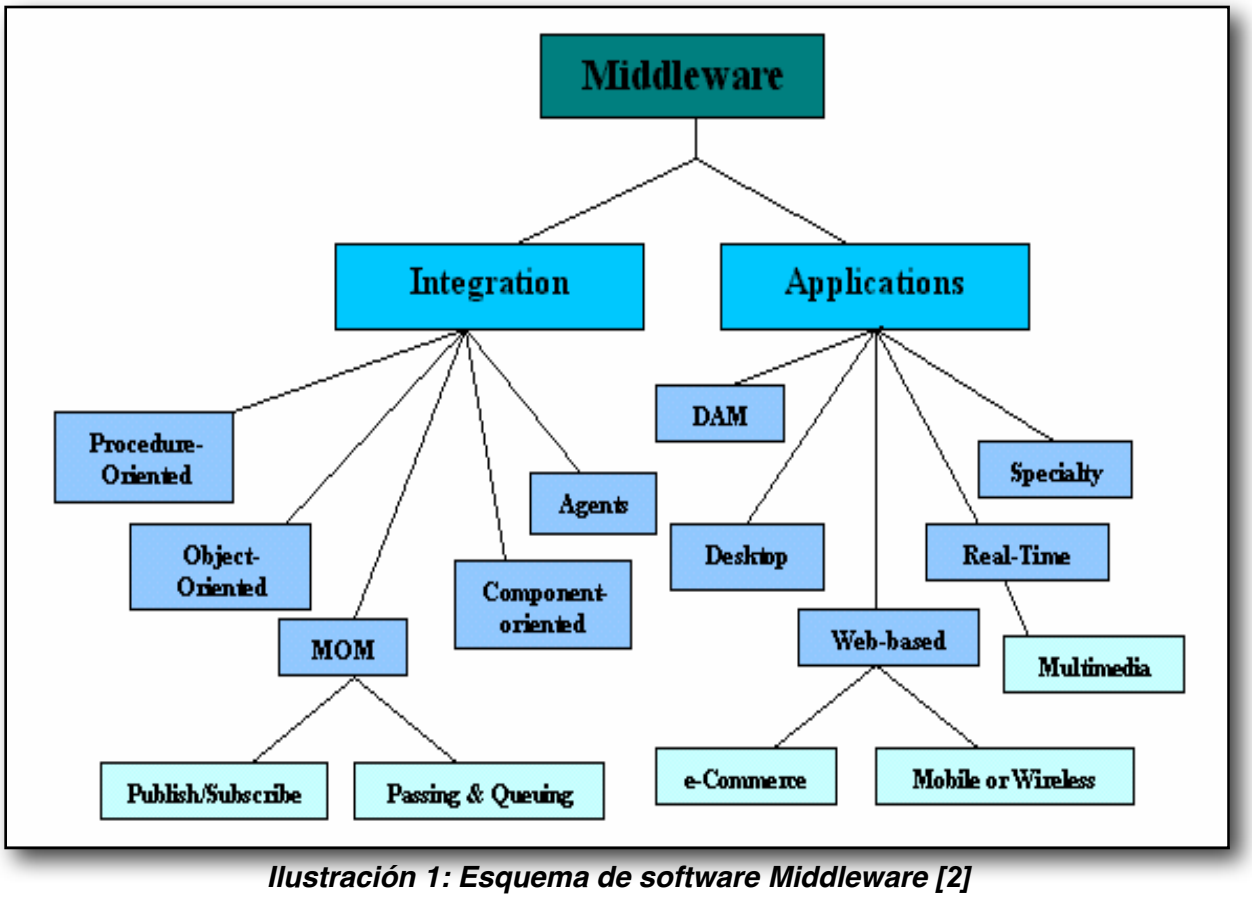


Ilustración 1: Esquema de software Middleware [2]

Debido a la heterogeneidad de sistemas y, en la mayoría de los casos, la incompatibilidad de los mismos, ha motivado a la generación del concepto *middleware*, que representa la opción más común para la construcción de sistemas informáticos distribuidos en la actualidad, como por ejemplo, sistemas de gestión de bases de datos, servidores web, servidores de aplicaciones, sistemas de gestión de contenido.

El *middleware* queda comprendido en un espacio entre el cliente y el servidor. A pesar de no pertenecer a ninguno de ellos, se ejecuta en ambas partes de la aplicación.

Esta capa intermedia se encarga de ofrecer a los usuarios la visión de un sistema único, ocultando la complejidad subyacente. De esta tarea se responsabilizan las partes más importantes de un *middleware*, el sistema operativo de red, *Network Operating System (NOS)*, y la pila de transporte, *Transport Stack*.

El *middleware,* haciendo uso del *NOS,* facilita al usuario los siguientes niveles de transparencia *[5]*:

- **Transparencia de localización:** permite acceder a un recurso sin conocer su localización exacta (en términos del nombre de la máquina en la que reside).

- **Transparencia de espacio de nombres:** permite utilizar la misma convención de nombres para acceder a cualquier recurso del sistema, con independencia del tipo de recurso o fabricante.

- **Transparencia de identificación:** permite utilizar la misma información de identificación para el acceso a todos los recursos del sistema.

- **Transparencia de acceso local o remoto:** permite utilizar cualquier recurso del sistema, remoto o no, como si fuera local.

- **Transparencia temporal:** permite obtener información temporal sincronizada con el resto de componentes del sistema.

- **Transparencia de errores:** permite ocultar errores en el acceso a ciertos recursos, por ejemplo mediante la redirección a un recurso redundante.

Los *middleware* se pueden categorizar según su capacidad de integración y según a la aplicación a la que se designan, además de por el uso al que estén destinados.

2.1.1 Categoría de integración

Gracias al tipo de integración que poseen los *middleware*, tienen la capacidad de unirse con sistemas heterogéneos, por ello poseen diferentes protocolos de comunicación o formas de operar en función del software al que están destinados. Estos *middleware*, como se mostró en la *Ilustración 1*, se pueden clasificar en cinco grandes tipos *[4]*, los cuales de explican a continuación: orientados a procedimiento o procesos, orientados a objetos y orientados a mensajes o agentes.

Orientados a procedimiento o procesos

Este tipo de *middleware* utiliza una comunicación sincronizada (como por ejemplo el teléfono). Una de las características de éstos, es que utilizan un *proxy* (programa o dispositivo que realiza una acción en representación de otro) cliente y un *proxy* servidor. El *proxy* cliente, convierte la petición en un mensaje que es mandado al servidor, posteriormente el *proxy* servidor recibe el mensaje, lo convierte en la petición y llama a la aplicación del servidor donde ésta es procesada. El *proxy* cliente chequea los errores, envía los resultados al software que inició la petición y entonces suspenden el proceso. Las ventajas de estos *middleware* es que usan un tipo estándar en nombres de servicios, pueden retornar respuesta aun con problemas en la red, además de manejar múltiples tipos de formatos para datos y niveles heterogéneos de sistemas de servicio. Las desventajas son que no poseen escalabilidad, no pueden retornar la información a un programa diferente del que realizó la solicitud (reflexión) y poseen procesos muy rígidos.

Orientados a objetos

La cualidad de este tipo de *middleware* es que soportan peticiones de objetos distribuidos. La comunicación entre los objetos puede ser sincronizada, sincronizada diferida o no sincronizada. Pueden soportar múltiples peticiones similares realizadas por varios clientes en una transacción. La forma de operar consiste en que el objeto cliente llama a un método lógico para obtener un objeto remoto. Un *proxy* local pone en orden la información y la transmite a través del agente (*broker*). El agente actúa como punto medio que contacta con un número de fuentes de información para obtener sus identificadores, recoger información y reorganizarla. El objeto servidor recibe el mensaje desde el agente y el *proxy* remoto recoge la información.

La información es ingresada a un objeto servidor. El resultado es devuelto de forma inversa. Las ventajas de este tipo de *middleware* es que permiten generar reflexión y escalabilidad. También opera con múltiples tipos de información y estados, además de soportar procesos múltiples.

Orientados a mensajes (MOM)

Este *middleware* se puede dividir en dos tipos, paso/espera y publicación/ suscripción. El "paso de mensaje" se inicia con el envío por parte de la aplicación de un mensaje a uno o más clientes, con el *MOM (Message Oriented Middleware)* del cliente. El servidor *MOM*, recoge las peticiones de la cola (*Message Broker*) en un orden o sistema de espera predeterminado. Los actos del servidor *MOM* son como un *router* y usualmente no interactúan con estas. El *MOM* de publicación y suscripción actúa de manera ligeramente diferente. Es más orientado a eventos. Si un cliente quiere participar por primera vez, se una al tópico de información. Dependiendo de su función, si es como publicador, suscriptor y ambas, éste registra un evento. El publicador envía unos datos al tópico, lo que produce un evento en el sistema. El servidor *MOM* recoge ese evento y envía los datos al subscriptor registrado cuando la información está disponible.

Orientados a componentes

Un componente es un "programa que realiza una función específica y está diseñado de tal manera que intercambia información fácilmente con otros componentes y aplicaciones". El *middleware* en este caso, es una configuración de componentes, es decir, un conjunto de programas configurados de una manera específica. Los puntos fuertes de este *middleware* es que es configurable y reconfigurable. La reconfiguración se puede realizar en tiempo de ejecución, lo que ofrece una gran flexibilidad para satisfacer las necesidades de un gran número de aplicaciones.

Agentes

Los agentes son un tipo de *middleware* que posee varios componentes: entidades, los medios de comunicación y las leyes. Las entidades pueden ser objetos o procesos, los medios pueden ser canales, tuberías, etc.

Las fortalezas de los *middleware* agentes son que pueden realizar una gran cantidad de tareas en nombre del usuario y que pueden cubrir una amplia gama de estrategias basadas en el entorno que les rodea. Sin embargo, su implementación es complicada debido a la complejidad y dificultades que se necesita para entender las operaciones que manejan.

2.1.2 Categoría según su aplicación

La clasificación por aplicación incluye los *middleware,* como se mostró en la *Ilustración 1*, que son específicos para un tipo de aplicación determinado *[4]*, los cuales se explican a continuación: aplicación DAM, *middleware* de escritorio, basados en web y *middleware* en tiempo real.

DAM

Los *middleware* para acceso a información *DAM (Data Access Middleware)*, tienen la característica de poder interactuar con diversas fuentes de datos. En este tipo de *middleware* se encuentran los que procesan transacciones, *gateways* de bases de datos y sistemas distribuidos de transacción/procedimiento. Las fortalezas que posee este tipo de *middleware* es la comunicación que tiene entre múltiples fuentes de datos, la conversión del lenguaje de programación de la aplicación a un lenguaje aceptado por la fuente de datos de destino y la capacidad de respuesta en un formato y lenguaje aceptable para el solicitante.

Middleware de escritorio

Los *middleware* de escritorio pueden hacer variaciones en la presentación de la información pedida por el usuario por aplicaciones de rastreo y asistencia, proveer una copia de seguridad de datos y otras operaciones de fondo. Otras de las operaciones pueden ser gráficas, ordenamientos, directorios de servicios, manejo de información de la base de datos, manejo de procesos, calendarización de trabajos, notificación de eventos de servicios, manejo de instalación de software, servicios de cifrado y control de accesos.

Middleware basados en la web

Este tipo de *middleware* asiste al usuario con la navegación web mediante el uso de una interfaz que le permiten encontrar páginas de su interés y detectar cambios de interés del usuario basado en su historial de búsquedas. Provee de un servicio de identificación para un gran número de aplicaciones y comunicación entre procesos independientes del sistema operativo. Los *middleware* que se encuentran fuertemente unidos a la red se llaman servidores de aplicaciones, ya que mejoran el rendimiento, disponibilidad, escalabilidad, seguridad, recuperación de información, y soportan la administración colaborativa y su uso.

Los *middleware* pueden contactar directamente con la aplicación consiguiendo así mejor comunicación entre el servidor y el cliente. Otros servicios importantes dados por este tipo de *middleware* son servicios de directorios, *email*, cadenas de suministros de gran tamaño, accesos remotos a información, descargar archivos, accesos a programas y acceso aplicaciones remotas.

Middleware en tiempo real

Los *middleware* en tiempo real soportan las peticiones sensibles al tiempo y políticas de planificación. Esto se realiza con servicios que mejoran la eficiencia de las aplicaciones de usuario. Los *middleware* en tiempo real se pueden dividir en diferentes aplicaciones (aplicación de base de datos en tiempo real, sensor de procesamiento y transmisión de información). La información que pasa a través de un *middleware* en tiempo real se ha incrementado dramáticamente con la introducción de Internet, redes inalámbricas, y las nuevas "aplicaciones basadas en la difusión".

Las fortalezas de este tipo de *middleware* son que proveen un proceso de decisión para determinar el mejor criterio para resolver procesos sensibles al tiempo y así ayudar a los sistemas operantes en la localización de recursos cuando tienen tiempos límites de operación. Los *middleware* multimedia es una rama mayor en los *middleware* en tiempo real, estos pueden manejar una gran variedad de información.

Estos tipos de información pueden ser textos, imágenes de todo tipo (*GPS (Global Positioning System)*, imágenes, etc.), procesadores de lenguajes naturales, música y vídeo. La información debe ser recopilada, integrada y entonces enviada al usuario sensible del tiempo. Los dispositivos multimedia pueden incluir una mezcla de dispositivos tanto físicos como lógicos.

2.1.3 Tipos de uso

Los *middleware* proporcionan un conjunto más funcional de la API para permitir a una aplicación realizar las siguientes acciones:

- Localizar claramente a través de la red, proporcionando así una interacción con otro servicio o aplicación.

- Los datos filtrados para que sean utilizables en un ambiente público a través de *anonymization process* para la protección de la privacidad

- Ser independiente del servicio de red

- Ser fiable y siempre disponible

- Añadir los atributos complementarios como semántica en comparación con el sistema operativo y servicios de red.

Middleware ofrece algunas ventajas únicas tecnológicas para los negocios y la industria. Por ejemplo, los sistemas tradicionales de bases de datos suelen ser desplegados en entornos cerrados, donde los usuarios acceden al sistema sólo a través de una red restringida o intranet (por ejemplo, red interna de una empresa).

Con el crecimiento de la *World Wide Web*, los usuarios pueden acceder a prácticamente cualquier base de datos para las que tengan derechos de acceso adecuados desde cualquier parte del mundo. El *middleware* aborda el problema de diferentes niveles de interoperabilidad entre las estructuras de base de datos diferente y facilita el acceso a la herencia de un *DBMS (Database Management System)* o aplicaciones a través de un servidor web, sin tener en cuenta las características específicas de base de datos *[5]*.

Las empresas frecuentemente utilizan las aplicaciones de *middleware* para vincular la información de bases de datos de sus departamentos, tales como nóminas, ventas y contabilidad, o bases de datos alojadas en múltiples localizaciones geográficas. En el mercado de la salud, altamente competitivo, los laboratorios hacen un amplio uso de aplicaciones de *middleware* para minería de datos y *LIS (Laboratory Information System)*, y para combinar los sistemas de información durante fusiones de hospitales. Los *middleware* ayudan a reducir la brecha entre *LISs* separados en una red de salud recién formada a raíz de una compra del hospital.

Los *middleware* pueden ayudar a los desarrolladores de software a no tener que escribir *APIs* para todos los programas de control, que actúa como una interfaz de programación independiente para sus aplicaciones. Para el futuro de Internet, el funcionamiento de la red a través del monitorización de tráfico en escenarios multi-dominio, utilizando herramientas de mediación (*middleware*) es una poderosa ayuda, ya que permiten a operadores e investigadores supervisar la calidad de servicio y analizar los eventuales fracasos en servicio de telecomunicaciones.

Por último, el comercio electrónico utiliza *middleware* para ayudar en el manejo de transacciones rápidas y seguras a través de muchos tipos diferentes de entornos informáticos.

En resumen, el *middleware* se ha convertido en un elemento crítico en una amplia gama de industrias, gracias a su capacidad de reunir los recursos a través de diferentes redes o plataformas de computación.

2.2 Sistemas distribuidos de datos

Un sistema distribuido de datos (DDS) es un *middleware* de tipo publicador-subscriptor en el que los computadores, conectados en red, comunican y coordinan sus acciones únicamente mediante el envío de mensajes. Es habitual que los sistemas distribuidos se caractericen por la concurrencia de los componentes, la ausencia de un reloj común y la independencia en los fallos. La concurrencia permite que los recursos disponibles, incluidos los propios componentes, puedan ser utilizados simultáneamente *[6]*.

La ausencia de un reloj común, o global, se debe a que la transferencia de mensajes entre los componentes no tiene una base temporal común sino que el factor temporal está distribuido entre los componentes. Finalmente, el aislamiento de los fallos de los componentes implica que cada componente debe tener una independencia con respecto al resto de manera que pueda seguir funcionando aunque otro componente no se encuentre disponible. Esto último se puede matizar en la medida en que un componente dependa de otro para su funcionamiento.

La coordinación de los componentes de un sistema distribuido se debe realizar para lograr varios objetivos para los que dichos componentes son necesarios. Para alcanzar dichos objetivos, los componentes deben realizar diversas tareas de manera independiente y servirse de forma concurrente de los recursos que tienen a su disposición.

La comunicación entre componentes obliga a que exista un intercambio de información entre los mismos. Este intercambio de información se debe realizar a través de los medios de comunicación que el sistema distribuido proporciona a cada componente y además implicará una topología de las comunicaciones. El acceso de los componentes a los medios de comunicación disponibles obliga a la existencia de uno o varios protocolos de comunicaciones, que deberían conocer los componentes que se comuniquen a través de los mismos.

El intercambio de información entre componentes supone que debe existir una manera de enviar la información de manera que sea comprensible por los agentes implicados en el intercambio. Consecuentemente el contenido de los mensajes debe disponer de un lenguaje, o al menos unas normas conocidas por los agentes que intervengan en el proceso de intercambio de información. La existencia de un lenguaje, supone la existencia de una gramática y consiguiente de una sintaxis. Estos aspectos del lenguaje de intercambio de información deben ser tenidos en cuenta a la hora de diseñar un sistema distribuido y de intentar lograr un alto grado de estandarización del mismo.

A continuación se van a comentar las características de los sistemas distribuidos de datos, profundizando en el paradigma de publicación/ suscripción y detallando finalmente las políticas de calidad de servicio que posee este tipo de *middleware*.

2.2.1 Características

Un sistema distribuido puede tener muchas propiedades deseables, las cuales enriquecerán el mismo haciéndolo más eficiente. Entre las propiedades deseables de un sistema distribuido se pueden destacar las siguientes *[7]*:

- **Heterogeneidad:** se entiende como la variedad y la diferenciación de los componentes de un sistema distribuido. Generalmente la variación de los diversos componentes de un sistema justifica la creación de estándares que permitan a los componentes conocer las reglas de funcionamiento con un esfuerzo mínimo de adaptación.

- **Extensibilidad:** este concepto es similar en algunos casos a la compatibilidad que el sistema puede ofrecer a nuevos componentes. La extensibilidad del sistema también justifica el uso de estándares, cuanto menos las "normas" del sistema deben ser conocidas por los componentes. También suele hablarse de sistemas abiertos cuando se refiere a sistemas extensibles.

- **Seguridad:** actualmente, es una de las propiedades que proporciona mayor valía a un sistema distribuido. Sobre seguridad hay tres aspectos que son importantes tener en cuenta: disponibilidad, confidencialidad e integridad.

    - **Disponibilidad:** los recursos del sistema están accesibles para aquellos componentes que lo requieran, en el instante en que sea necesario.

    - **Confidencialidad:** protección que el sistema proporciona para no ser accesible por componentes no autorizados. Además, debe proporcionar la suficiente seguridad a un componente, que los componentes con los que está trabajando son realmente los que él requería para trabajar, es decir, además de evitar la intrusión, un sistema confiable debe proporcionar suficientes garantías para evitar la suplantación.

- **Integridad:** protección que el sistema suministra contra la alteración, ya sea accidental como provocada, de los datos que circulan por el sistema.

- **Escalabilidad:** un sistema distribuido es escalable en la medida en que se pueden aumentar sus componentes sin que esto conlleve una pérdida de efectividad debida a los cambios. El objetivo final de un sistema, con respecto a la escalabilidad, es que apenas deben realizarse cambios en el mismo pese al aumento de los componentes. Normalmente es un gran logro que un sistema que se adapta a las necesidades de los componentes cuando la cantidad de estos cambia, no requiera de modificaciones.

- **Soporte a fallos:** los componentes de un sistema distribuido pueden fallar y es normal que esos fallos antes o después se produzcan. Un fallo en uno o varios componentes hará que el resultado del sistema sea diferente o de menor calidad que el esperado, por ello la prevención y tratamiento de los fallos es un aspecto que deberá tener muy en cuenta un sistema distribuido. Algunas características que deben tener los sistemas distribuidos con respecto a los fallos son:

  - **Detección:** debe realizarse lo antes posible para evitar que los errores en el procesamiento tengan consecuencias colaterales en el resto de los componentes.

  - **Enmascaramiento:** una vez detectado el fallo, éste debe enmascararse o atenuarse, es decir que aunque el fallo es conocido, los componentes no se vean afectados.

  - **Tolerancia:** implica que los componentes conozcan el fallo y reaccionen frente al mismo de una manera conocida y controlada.

  - **Recuperación:** es la fase en la que el sistema, una vez solucionado el fallo, vuelve a funcionar según los requerimientos.

- **Concurrencia:** la concurrencia es la capacidad que tiene un sistema para que un recurso, pueda ser utilizado simultáneamente. Realmente la simultaneidad puede no ser completa, sino a efectos de los componentes que acceden al recurso. La concurrencia supondrá el uso de diversas estrategias de acceso al recurso con el objetivo de mantener la integridad de la información.

- **Transparencia:** la transparencia de un sistema es la ocultación al usuario de los detalles que son propios de la gestión interna del sistema. De esta manera, los usuarios de un sistema no lo perciben con más detalles que el estrictamente necesario. La transparencia se logra de diversas maneras, pero fundamentalmente a partir del encapsulado de componentes y del uso de interfaces.

A medida que un sistema distribuido dé soporte a las propiedades anteriores, sus componentes, y consiguientemente el sistema en su conjunto, verán aumentada su eficiencia e incrementado su valor. Para poder proporcionar estas características, se deberá realizar un esfuerzo considerable, tanto en el diseño, como en la monitorización, gestión y control del sistema, por lo que estos aspectos deberán ser tratados con especial atención en el desarrollo del sistema.

2.2.2 Paradigma Publicación-Suscripción

El paradigma de la publicación-suscripción se puede implementar haciendo uso tanto de arquitecturas centralizadas como descentralizadas o híbridas. Tomando, por ejemplo, el modelo cliente-servidor, cada vez que el cliente necesita alguna información activamente debe hacer una petición al servidor. Éste es un buen modelo para muchas situaciones, pero en otras este método es poco eficiente. Pensemos, por ejemplo, en el caso de una agencia de bolsa que quiere mantener informados a sus clientes de la evolución en tiempo real de las cotizaciones de las acciones; o el caso de una agencia de noticias que distribuye información al momento. En estos casos, los receptores tendrían que ir consultando continuamente el servidor para tener la última cotización o la última noticia, con la sobrecarga que esto significa tanto a la red como al cliente y al servidor *[24]*.

La manera como el paradigma publicación-suscripción aborda estas situaciones, como se muestra en la *Ilustración 3*, es haciendo que un productor de información anuncie la disponibilidad de un cierto tipo de información, un consumidor interesado se suscriba a esta información y el productor periódicamente vaya publicando información.

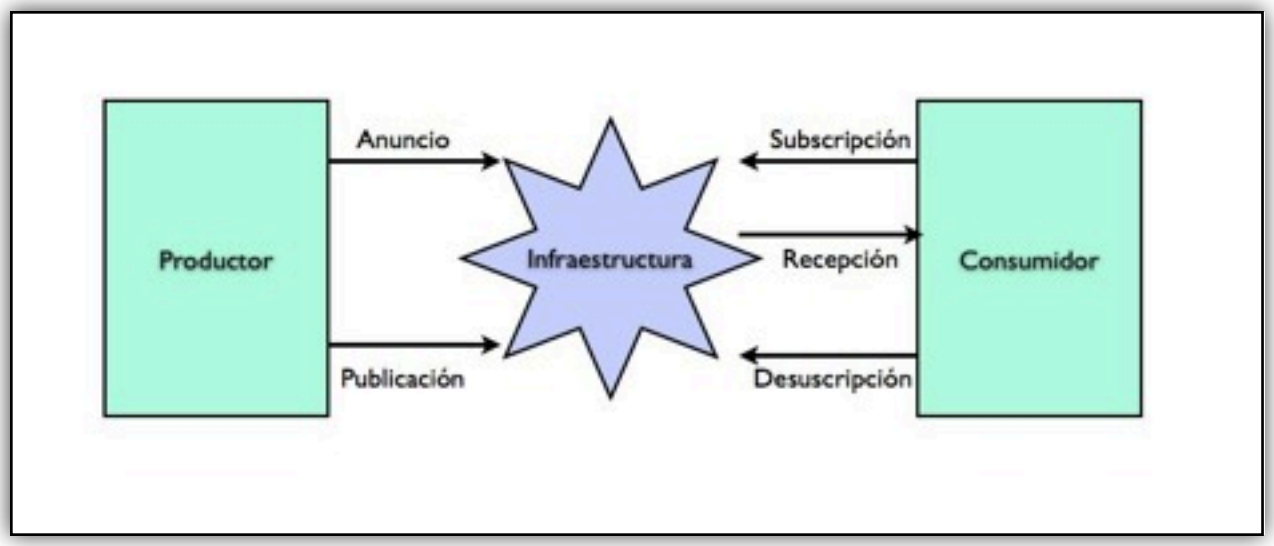


Ilustración 3: Modelo Publicador - Subscriptor [24]

Para poder tener el comportamiento descrito, la arquitectura publicación-suscripción está formada por los siguientes componentes *[24]*:

- **Productor de información (*publish*):** aplicación que tiene la información que hay que difundir. El productor publica esta información sin tener que saber quién está interesado en recibirla. Envía la información a través de canales.

- **Consumidor de información (*subscribe*):** aplicación interesada en recibir información. El consumidor se suscribe a los canales que diseminan la información que le interesa. Recibe esta información por los canales a los que está suscrito.

- **Mediador (*DomainParticipant*):** está entre el productor y el consumidor de información. Recibe información de los productores y peticiones de suscripción de los consumidores. También se encarga de encaminar la información publicada a los destinatarios suscritos al canal. Este mediador puede estar distribuido. En este caso, es necesario que los diferentes mediadores se organicen para proveer los canales.

- **Canal:** son los conectores (lógicos) entre los productores y los consumidores de información. El canal determina varias de las propiedades de la información que hay que diseminar y de las funcionalidades soportadas: tipo de información, formato de los datos, posibilidades de personalización del canal por parte del usuario (por ejemplo, selección de contenidos, modos de operación), si el contenido expira o es persistente, estrategia que se seguirá para hacer las actualizaciones, si los datos se entregan sólo una vez o si, en cambio, garantizamos que se puede recibir el contenido independientemente del momento en el que se generó, modo de operación (si se da apoyo por el modo de operación en desconectado), pago (cuál es la política de pago que se utiliza: pagar por ver, por tiempo, por contenido, etc.).

  Los canales modelan una relación de uno a muchos entre productores y consumidores. Habitualmente también son necesarios canales para que los consumidores se puedan relacionar de uno a uno con los productores. Esto se acostumbra a hacer en un estilo cliente-servidor y, por lo tanto, desde el punto de vista conceptual estaría fuera del sistema publicación-suscripción.

- **Tópico (*Topic*):** describen los datos asociados a un tópico y se usan para asociar los lectores a los escritores. Se identifica mediante una cadena de texto a la que luego se le enlaza un tipo de datos. Se describen por recomendación en *IDL*.

- **Escritor (*DataWriter*):** construcción que permite el acceso para escribir datos tipados sobre un tópico particular.

- **Lector (*DataReader*):** construcción que permite el acceso para leer datos tipados relativos a un tópico específico.

Como resumen, cuando una aplicación desea publicar una información, lo realiza escribiendo en un tópico, a través de un objeto publicador. Este objeto reside en un dominio de participación que gestiona tanto los objetos publicadores como los subscriptores o escuchadores, que son los responsables de recibir los mensajes. Los objetos publicadores proporcionan los mensajes a petición de la aplicación, mientras que los objetos subscriptores avisan a la aplicación de la llegada de un mensaje. En este último caso la iniciativa de la comunicación la toma el sistema de comunicaciones y no la aplicación.

En la *Ilustración 4* se observa la relación entre los distintos componentes de una arquitectura publicación-suscripción.

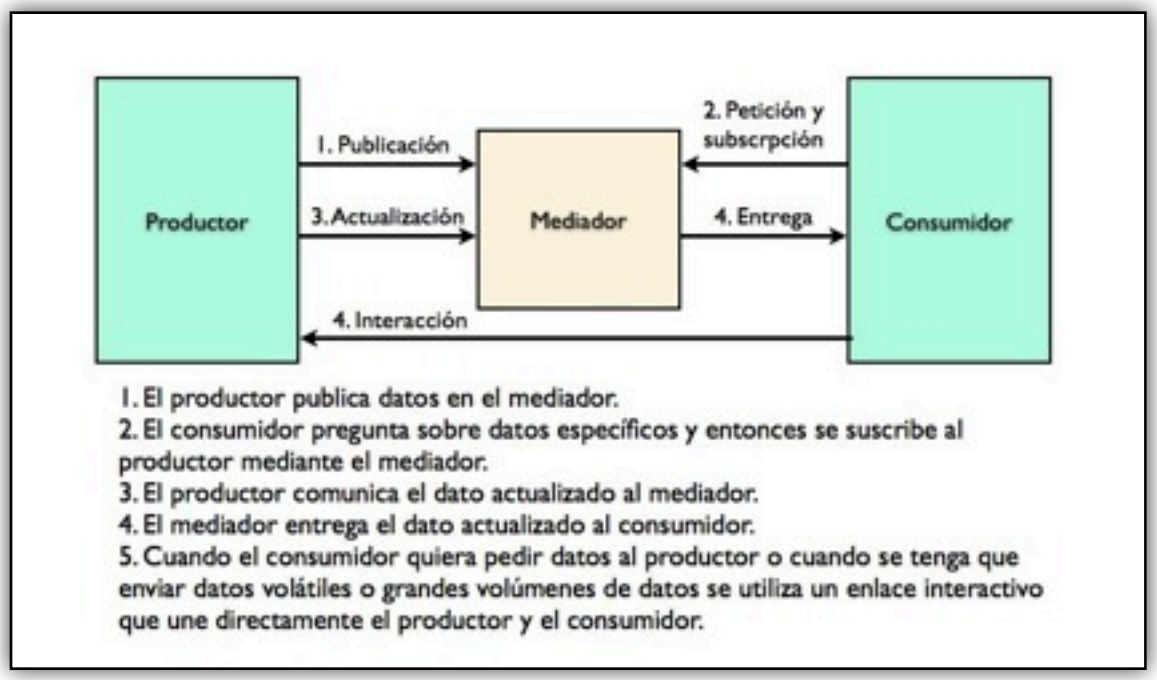


Ilustración 4: Arquitectura publish-subscribe [24]

Tal y como se ha visto, los sistemas publicación-suscripción permiten una distribución asíncrona de información. A continuación se indican algunas situaciones y aspectos en los que estos sistemas pueden ser una alternativa apropiada *[9]*:

- **Localización:** para los usuarios es un problema saber dónde está la información que les interesa.

Aunque haya buenas herramientas de búsqueda, muchas veces la información obtenida no es de la calidad deseada. En los sistemas publicación-suscripción, el usuario se suscribe a unos canales y ahora es el proveedor de información quien asume el rol activo de hacer llegar su información a los interesados.

- **Focalización:** puesto que el usuario dice explícitamente cuáles son sus preferencias, es fácil proporcionar la información centrada en sus intereses.

- **Personalización:** el usuario puede especificar que, antes de que los datos y sus propiedades se entreguen, se apliquen ciertos requerimientos. Por ejemplo, formato de los datos, prioridad, palabras clave, etc.

- **Actualidad:** los datos se pueden diseminar a medida que están disponibles. El proveedor de información puede invalidar los datos obsoletos.

- **Adaptación (*tailoring*):** el proveedor también puede decidir qué información ve el receptor y cuál no.

- **Reducción del tráfico:** puesto que el sistema disemina la información a quien está interesado en recibirla, se reduce mucho el tráfico en la red. Intentar localizar la información puede provocar mucho tráfico. Además, si se utiliza una infraestructura de transporte apropiada, aún se puede reducir más la ocupación de la red.

Las arquitecturas publicación-suscripción están pensadas para proporcionar tres tipos de servicios: coordinación de procesos, replicar contenidos e informar a personas.

Algunos de los campos en los que se utilizan aplicaciones publicación-suscripción son los siguientes *[9]*:

- **Grupos de noticias y listas de distribución de correo:** los mensajes Usenet y las listas de distribución de correo se pueden considerar como sistemas de publicación-suscripción un poco primitivos.

Por ejemplo, los mensajes Usenet diseminan artículos por todo Internet. Un servidor de mensajes se suscribe a otros servidores de mensajes y recibe los mensajes de los grupos a los cuales está suscrito. Cuando en un grupo se genera un nuevo artículo, el servidor en el que se ha generado el artículo se encarga de que este artículo se disemine hacia otros servidores.

- **Bolsa y noticias:** los sistemas que informan sobre la evolución de las acciones en la bolsa o las agencias de noticias son otro ejemplo de sistemas publicación-suscripción. En estos sistemas, los usuarios especifican unos intereses y el sistema debe garantizar que los usuarios dispongan en todo momento de la información tan actualizada como sea posible.

- **Sistemas de información de tráfico:** como en las aplicaciones para la bolsa y las noticias, es necesario que la información se envíe la mayoría de las veces en tiempo real. La información también se distribuye por medio de ordenadores o dispositivos móviles.

- **Distribución de software:** muchos sistemas requieren que el software se actualice frecuentemente. Por ejemplo, es necesario que el software de los bancos de inversiones, por necesidades de seguridad, se actualice frecuentemente y extensivamente. Utilizando una aplicación publicación-suscripción se consigue que el sistema esté funcionando continuamente y actualizado a la última versión sin problemas de seguridad para las actualizaciones.

2.2.3 Calidad de servicio

Para lograr que un sistema distribuido cumpla unos requisitos adecuados, o lo que es lo mismo, que logre cumplir las características que se le requieren, es necesario determinar unos parámetros en los que basar las características. Estos parámetros, deben ser medibles y representativos. En los sistemas distribuidos basados en componentes que ofrecen unos servicios a las aplicaciones que sobre ellos funcionan, los parámetros que se miden y permiten determinar el buen funcionamiento del sistema, se conocen como parámetros de calidad de servicio *[6]*.

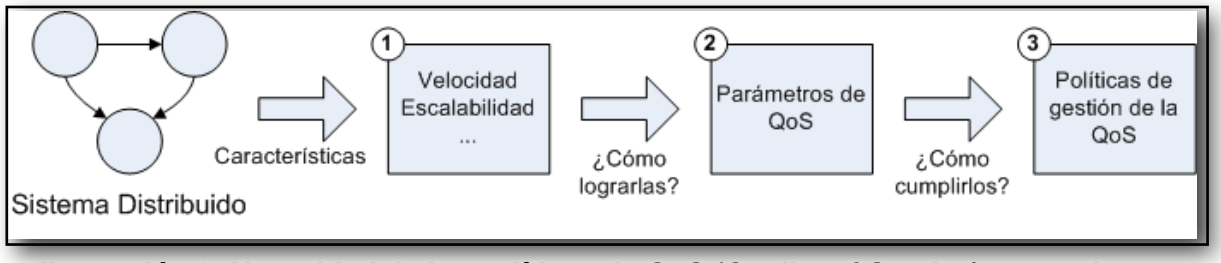


Ilustración 2: Necesidad de las políticas de QoS (Quality of Service) en un sistema distribuido [6]

Los parámetros de calidad de servicio proporcionan un medio por el que poder tomar decisiones acerca de los componentes, y de los mismos servicios, del sistema. Para poder cumplir esos requisitos se debe gestionar los parámetros entre los componentes. La gestión, incluida la negociación, de los parámetros de calidad de servicio entre los componentes se realiza por medio de políticas de calidad de servicio.

A continuación se muestra la lista con los parámetros de calidad de servicio más utilizados en *DDS*. Para ello, se hace uso de la *Tabla 1*, donde la primera columna es el nombre del parámetro de calidad del servicio, la segunda columna indica mediante abreviaturas a qué participante afecta el parámetro, la tercera columna indica si el parámetro es producto de la negociación entre participantes (un ejemplo de esto se produce cuando el lector quiere datos ordenados y el escritor no acepta esta petición, ofreciéndolos no ordenados) y la última especifica si el parámetro es modificable tras la creación de la política de calidad de servicio o no *[8]*.

Las siglas utilizadas para determinar al participante son: Participante de Domino (DP), Publicador (P), Suscriptor (S), Tópico (T), Lector (DR) y Escritor (DW).

QOS POLICY	CONCERNS	RXO	CHANGEABLE
User data	DP, DR, DW	NO	NO
Group data	P, S	NO	YES
Topic Data	T	NO	YES
Durability	T, DR, DW	YES	NO
Presentation	P, S	YES	NO
Deadline	T, DR, DW	YES	YES
Latency Budget	T, DR, DW	YES	YES
Ownership	T	YES	NO
Ownership Strength	DW	N/A	YES
Entity Factory	DP, P, S	NO	YES
Lifespan	T, DW	N/A	YES
Liveliness	T, DR, DW	YES	NO
Time Based Filter	DR	N/A	YES
Partition	P, S	NO	YES
Reliability	T, DR, DW	YES	NO
Transport Priority	T, DW	N/A	YES
Destination Order	T, DR	NO	NO
History	T, DR, DW	NO	NO
Resource Limits	T, DR, DW	NO	NO
Reader Data Lifecycle	DR	N/A	YES
Writer Data Lifecycle	DW	N/A	YES

Tabla 1: Parámetros de QoS [8]

- **Plazo (*Deadline*):** este parámetro es aplicado al tópico, al lector y al escritor[1]. Indica la velocidad mínima a la que el escritor enviará los datos. Es un parámetro útil cuando se debe enviar periódicamente un dato. También, por lo tanto, indica el máximo tiempo que el lector está dispuesto a esperar por nuevos datos.

[1] Cuando un parámetro de calidad de servicio se refiere a un periodo de publicación, recepción, etc., en realidad lo que sucede es que el escritor notifica a su publicador el periodo con el que quiere que los datos que le envía sean transmitidos por la red.

- **Orden en el Destino (*Destination Order*):** este parámetro es aplicado al tópico y al lector. Permite decidir, si la entrega al destinatario se hace en el orden de recepción o de envío. Para tomar esta decisión se apoya en el uso de *Timestamps* o marcas de tiempo que los datos tienen asociados.

- **Durabilidad (*Durability*):** este parámetro es aplicado al tópico, al lector y al escritor. Determina cómo se almacenan los datos. Existen tres posibles formas de almacenamiento:

    - **Datos volátiles (*volatile*):** es la opción por defecto. Los datos antiguos son eliminados, cuando nuevos datos son recibidos.

    - **Datos temporales (*transient*):** en este caso, el sistema mantendrá un número finito de datos marcado por el programador en memoria volátil.

    - **Datos persistentes (*Persistent*):** el sistema mantendrá en la memoria todos los datos que se vayan recibiendo de forma permanente. Esto permite que cualquier suscriptor pueda unirse al sistema con la misma información que los demás puesto que un escritor le enviará todos los datos anteriores a la creación del lector.

- **Creación de entidades (*Entity Factory*):** este parámetro se aplica al participante de dominio, al suscriptor y publicador. Todas las entidades deben ser activadas antes de que sean visibles en la red y usadas para la comunicación. Mediante este parámetro un usuario puede indicar qué entidades se activan automáticamente y cuáles deben ser supervisadas por el administrador.

- **Datos de grupo (*Group Data*):** este parámetro se aplica al publicador y suscriptor. Consiste en información extra que es añadida a un tópico que puede ser accedido por una aplicación cuando una nueva entidad es descubierta en el dominio. Su uso habitual permite al diseñador del sistema la adjudicación de políticas a nivel de aplicación. También podría ser útil para autenticación y autorización.

- **Historia (*History*):** este parámetro aplica a tópicos, escritores y lectores. Especifica cuántos datos serán almacenados por la infraestructura *DDS*. Por defecto se mantiene 1. Es muy útil para que las nuevas aplicaciones que se unan a un tópico obtengan datos antiguos del mismo.

- **Latencia recomendable (*Latency Budget*):** este parámetro es aplicado al tópico, lectores y escritores. Básicamente establece por parte de los suscriptores hacia los publicadores un tiempo máximo aceptable de retardo en la entrega de mensajes. Se utiliza para asignar prioridades en la transmisión de mensajes.

- **Tiempo de vida (*Lifespan*):** este parámetro es aplicado sobre tópicos y escritores. Establece la máxima duración de la validez de cualquier mensaje individual. Cuando un dato es publicado por un escritor, el *lifespan* marca cuánto tiempo tiene de validez para el resto de participantes.

- **Modo de vida (*Liveliness*):** este parámetro es aplicado sobre los tópicos, lectores y escritores. Especifica cómo la infraestructura *DDS* determina si una entidad está viva o no. Tiene dos sub-parámetros: el mecanismo usado para determinar el tiempo de vida y una duración de cuánto tiempo el escritor ejecuta la señalización de que está vivo. Puede tomar tres valores:

    - **Automático (*Automatic*):** un mensaje de *alive (vivo)* es enviado automáticamente tan a menudo como se indique. Por defecto es infinito.

    - **Manual por participante (*Manual by participant*):** si las otras entidades participantes del dominio están activas, la entidad también se considera activa.

    - **Manual por tópico (*Manual by topic*):** se considera que el escritor está vivo si al menos tiene un dato de la aplicación.

- **Dueño (*Ownership*):** este parámetro es relativo al tópico. Especifica si diferentes escritores pueden actualizar la instancia de un objeto. Por defecto es posible. Se utiliza para dar redundancia de fallos (dos escritores para un mismo dato).

- **Importancia del dueño (*Ownership strength*):** parámetro que se adjudica a los escritores y establece un número con la importancia de sus datos. Esto permite que unos escritores tengan prioridad para escribir datos sobre otros.

- **Partición (*Partition*):** este parámetro es aplicado a publicadores y suscriptores. Es un modo de separar tópicos dentro de un dominio. Su valor es una cadena de texto y sólo recibirá mensajes de otros publicadores con la misma cadena de texto. Por defecto está vacío, lo que significa que no se realiza esta comprobación.

- **Presentación (Presentation):** parámetro que se aplica a publicadores y suscriptores. Determina cómo los datos son presentados a la aplicación cuando se trata con un suscriptor que tiene un grupo de lectores asociados. Tiene tres sub-parámetros:

    - **Acceso coherente (*Coherent Access*):** mantiene la estabilidad de los datos.

    - **Acceso ordenado (*Ordered Access*):** controla el orden de los cambios.

    - **Alcance del acceso (*Access Scope*):** aporta granularidad a los cambios en tres niveles: instancia, tópico o grupo. Este último se refiere al conjunto de lectores o escritores que pertenecen a un mismo suscriptor o publicador.

- **Ciclo de vida del lector (*Reader Data Lifecycle*):** especifica el tiempo que un lector mantendrá un dato desde que no exista el escritor para dicho dato. Es decir, si todos los escritores para un determinado dato se han caído o han dejado de producir nuevos datos, especifica cuánto tiempo se mantendrán esos datos. Por defecto se mantiene indefinidamente.

- **Tipo de entrega (*Reliability*):** este parámetro es aplicado al tópico, lector y escritor. Especifica si el dato a recibir será retransmitido en caso de error (*Reliable-fiable*) o no (*Best effort- mejor esfuerzo*).

- **Límite de Recursos (*Resource Limits*):** este parámetro se aplica al tópico, al lector y al escritor. Especifica cuanta memoria local se puede usar en la infraestructura *DDS*. Además, permite establecer el máximo número de instancias por tópico, el máximo número de datos por tópico y el máximo número de datos por instancia.

- **Filtro temporal (*Time-Based Filter*):** parámetro que configura el lector y establece un periodo mínimo de separación entre nuevos mensajes. Por defecto es cero lo que significa que no se limita este tiempo. De este modo se ayuda a no saturar a lectores de pocos recursos.

- **Datos del tópico (*Topic Data*):** este parámetro especifica si hay información adicional con respecto a un tópico.

- **Prioridad de transporte (*Transport Priority*):** este parámetro aplica a los tópicos y a los escritores. Especifica a los protocolos de transporte de niveles inferiores cómo de prioritario es un dato si dichos protocolos tienen capacidad para establecer diferentes prioridades.

- **Datos de usuario (*User data*):** parámetro que se puede aplicar a un tópico, lector y escritor. Permite añadir algún tipo de información extra sobre una entidad.

- **Ciclo de vida del escritor (*Writer Data Lifecycle*):** parámetro que controla la renovación automática de datos desde la cola de un escritor que ha dejado de estar registrado recientemente.

2.3 Multimedia

Multimedia se puede definir como *[10]* el sistema o aparato que utiliza conjunta y simultáneamente diversos medios, como imágenes, sonidos y texto, en la transmisión o reproducción de una información. Es un concepto tan antiguo como la comunicación humana, en la que se suelen combinar mensajes orales (sonido) junto a gestos del interlocutor (imagen). La conjunción de varios elementos transmisores del mensaje mejora la calidad de la transmisión de conceptos, a pesar de que este proceso implique siempre una pérdida semántica, aunque sea mínima.

La presentación de la información en varios formatos no es, por tanto, nada nuevo, pero el término multimedia suele denotar la presentación de la información en varios formatos digitales. Cuando un programa de computador o un documento digital combina adecuadamente varios medios, se mejora notablemente la atención, la compresión y el aprendizaje, ya que se aproxima más a la manera natural en que los seres humanos se comunican. Existen numerosos estudios acerca de la comunicación y del papel del signo (tanto verbal como no verbal), campo que en el mundo de la lingüística se cataloga con el término semiótica *[11]*.

La información contenida en un elemento multimedia está, normalmente, comprendida dentro de algunos de estos tipos básicos:

- **Texto:** en cualquier presentación, con o sin formato, e hipertexto.
- **Gráficos:** representación de esquemas, planos, dibujos lineales, gráficas.
- **Imágenes:** documentos formados por píxeles, obtenidos por copia del entorno (mediante scanner o cámara fotográfica) o sintéticamente.
- **Animación:** presentación de una secuencia de gráficos, que generan en el observador la sensación de movimiento.
- **Vídeo:** presentación de una secuencia de imágenes, que generan en el observador la sensación de movimiento.
- **Sonido:** voz, música o cualquier otro media perceptible por el oído.

Los contenidos multimedia forman, cada día más, parte de la vida cotidiana, empresarial e industrial. Se han ido introduciendo en áreas tan dispares como el arte, la educación, la medicina o la ingeniaría. Mención aparte merece la industria del entretenimiento, cuya actividad está basada caso por completo en el uso de contenidos y tecnologías multimedia.

Esta popularización del concepto multimedia ha dado lugar a una gran implantación de accesorios electrónicos personales que permiten disfrutarlo: desde reproductores musicales portátiles, hasta cámaras fotográficas o de vídeo compactas y accesibles a todos los bolsillos, sin olvidar los teléfonos de última generación o *smartphones* que aúnan todas estas funcionalidades en un único aparato.

A continuación, se abordará el *streaming* como método de transmisión de contenidos multimedia, ya que es la forma en la que se quiere implementar la aplicación final, así como de los diversos contenedores y códecs existentes, centrándonos finalmente en Mpeg-2, ya que es el más utilizado para este tipo de transmisiones.

2.3.1 *Streaming*

El proceso de *streaming* consiste en la entrega de uno o varios medios multiplexados hacia un cliente en tiempo real, usando una red con un determinado ancho de banda. En el proceso de *streaming* no hay ningún fichero que se descarga al ordenador del cliente, sino que el medio se reproduce conforme se está recibiendo, y a su vez el medio se recibe a la velocidad adecuada para su reproducción. Esto contrasta con las descargas progresivas, en las que el fichero sí queda descargado en disco y además se recibe a la mayor velocidad posible, con el fin de terminar el proceso de descarga lo antes posible *[12]*.

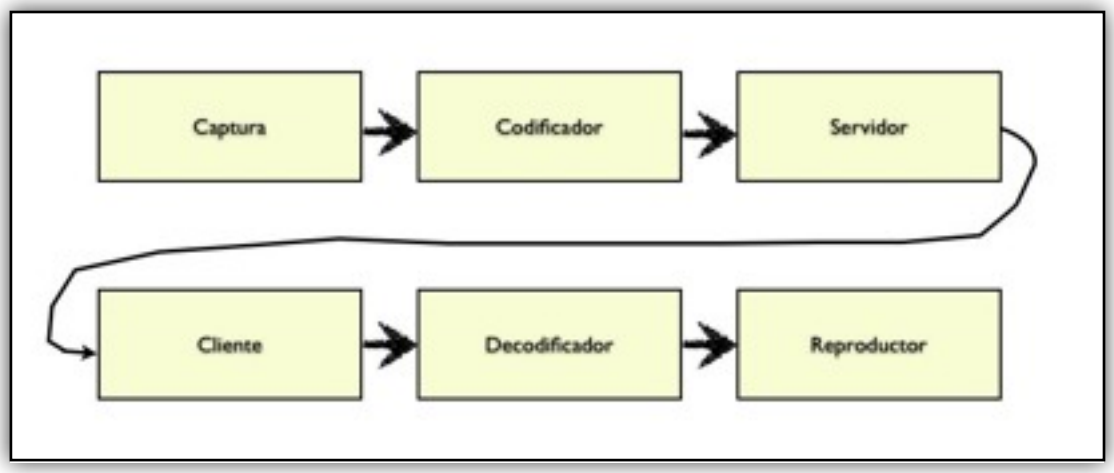


Ilustración 5: Etapas del proceso de Streaming [13]

Como se muestra en la *Ilustración 5*, las principales etapas involucradas en el proceso de *streaming* son *[13]*:

- **Captura:** el contenido audiovisual es capturado y convertido en datos digitales crudos, aún sin codificación y compresión.
- **Codificador:** los datos crudos de la etapa anterior que contienen gran cantidad de redundancia espacial y temporal, son codificados por algún códec específico y son encapsulados por algún formato contenedor particular.
- **Servidor:** el flujo continuo de datos es distribuido por la red a los diferentes clientes ya sea a través de *unicast*, *multicast* o *broadcast*.
- **Cliente:** en esta etapa se reciben los datos desde la red y se envían como un flujo a la etapa de decodificación.
- **Decodificador:** en esta etapa los datos son decodificados y descomprimidos de acuerdo al formato contenedor y a los códecs usados en el proceso de codificación.
- **Reproductor:** esta es la etapa en que el contenido multimedia, regenerado a partir de los datos, es desplegado en pantalla y reproducido por la tarjeta de sonido del receptor.

En un proceso de *streaming* estándar de audio y vídeo sincronizado, las peticiones de servicio por parte de los clientes se pueden manejar utilizando el protocolo *RTSP (Real-Time Streaming Protocol)*. Este protocolo se encarga de controlar el flujo de contenido multimedia en dos direcciones, de forma que los clientes pueden pedir al servidor hacer cosas como rebobinar la película, saltar al siguiente capítulo, etc. Esto se puede conseguir con *streaming* ya que el medio no se descarga linealmente sino que se reproduce conforme se obtiene, y se permiten saltos en la reproducción, consiguiendo un acceso aleatorio al medio, incluso en saltos hacia delante.

Por otra parte, los datos del medio (el *stream* que contiene típicamente audio y vídeo sincronizados) se pueden transportar usando el protocolo estándar *RTP (Real-Time Transport Protocol)*, que es un protocolo de transporte que permite la transmisión de información multimedia en tiempo real sobre cualquier tipo de red (aunque su uso más habitual es sobre redes usando el protocolo *UDP (User Datagram Protocol)*).

Tipos de *streaming*

El proceso de *streaming* se puede dividir en dos categorías, en función de cómo se obtiene la información a difundir *[12]*:

Streaming en directo

Es aquel que transmite eventos que están sucediendo justo en el momento de la difusión. Por ejemplo, la transmisión de conciertos o de clases son eventos que típicamente se difunden usando este tipo de *streaming*. La transmisión de radio y televisión por Internet también tiene estas características, aunque en ocasiones parte de la información que se difunde no parte de un evento en directo (por ejemplo, un programa que ha sido grabado previamente, pero que se va a difundir en un momento determinado).

En este tipo de transmisión se emplea el término difusión (*broadcast*) porque realmente se está transmitiendo "en vivo" a todos los clientes la misma información, que no es más que el evento que se está produciendo en ese momento. Así, independientemente de cuándo se conecta un cliente al servidor, todos ven exactamente el mismo punto del *stream* en un instante determinado (excepto las lógicas variaciones de los retardos en la red que hacen que unos clientes reciban antes los datos que otros). Para poder efectuar este tipo de transmisión no es suficiente con disponer de un servidor de *streaming*, sino que también es necesario un equipo que realice el proceso de captura y compresión en tiempo real.

Este equipo puede estar instalado en la misma máquina que el servidor de *streaming* si el número potencial de clientes no es grande, pero para resultados profesionales, en un entorno con muchos clientes, es conveniente separar ambos programas en dos máquinas distintas. Además, para dar un servicio realmente eficiente de este tipo de *streaming* es conveniente que la difusión se realice con técnicas de *multicast*.

Streaming multimedia bajo demanda

La transmisión del medio empieza desde el inicio del evento a ser reproducido para cada uno de los clientes. El medio a transmitir puede estar ya preparado desde el comienzo del proceso en un fichero comprimido. En este caso no representa una ventaja adicional el disponer de posibilidad de realizar *multicast* en la red, ya que cada cliente recibe una parte distinta del *stream* y por lo tanto un paquete de datos diferente.

Difusión mediante multicast

En *multicast*, un único *stream* se comparte entre diferentes clientes, de forma que el servidor envía la información una única vez, y ésta llega a todos los clientes que han demandado el servicio. Como ya hemos comentado, esta técnica es ideal para la difusión de medios en vivo (por ejemplo, radio en Internet) ya que todos los clientes están dispuestos a recibir el mismo flujo de datos (por ejemplo, el audio que corresponde al programa de radio que una emisora está difundiendo en un momento determinado) *[12]*.

Esta técnica reduce claramente el tráfico en la red, y evita posible congestión en los *routers*. Sin embargo, para llevarla a cabo es necesario tener acceso a una troncal con soporte *multicast*, o que el servidor y los clientes estén conectados a una red o redes *IP (Internet Protocol)* bajo un mismo dominio de administración en las que el *multicast* esté habilitado y existan *routers* dispuestos a encaminar información *multicast*.

En la transmisión típica *unicast*, cada cliente inicia su propio *stream*, independientemente de que todos los clientes estén interesados en el mismo *stream* de datos (esto es, aunque sea una difusión "en vivo"), de forma que se inician muchas conexiones uno-a-uno (una entre el servidor y el cliente por cada uno de los clientes).

Una solución para realizar una difusión en vivo de un *stream* multimedia es el caso en el que todos los clientes se encuentran conectados a una misma red *IP* con soporte *multicast*, y el equipo de difusión que realiza la captura se encuentra situado en otra red distinta (por ejemplo, la difusión de un mensaje de una compañía a una filial geográficamente separada). En este caso, la transmisión se puede hacer mediante *unicast* entre el equipo de difusión y la red en la que se encuentran los clientes, y aplicar una difusión *multicast* en la red destino.

El esquema básico para la definición de *streaming* se presenta en la *Ilustración 6*, en donde se puede observar una red habilitada *multicast* compuesto por un servidor y múltiples clientes que hacen requerimientos de objetos multimedia los cuales pueden ser mostrados a medida que se reciben.

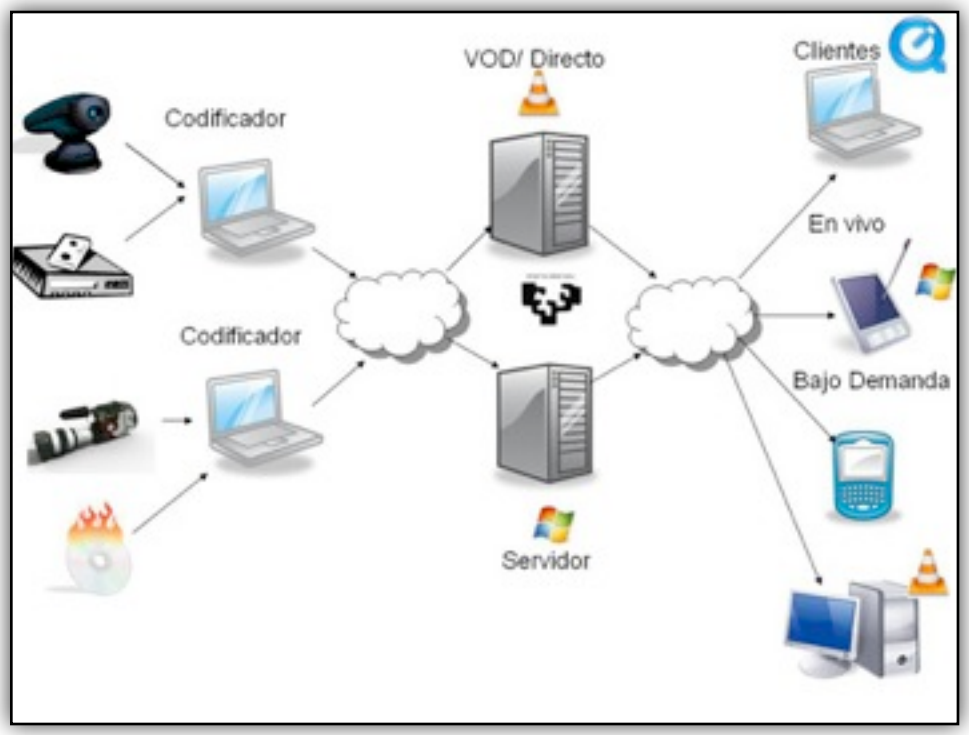


Ilustración 6: Esquema básico de Streaming [23]

Las etapas básicas de un proceso de *streaming*, como ya se comentó anteriormente, son: la captura de la información multimedia, codificación, inserción de la información dentro del servidor, la transmisión de los archivos al cliente previo establecimiento de conexión y la reproducción por parte del cliente.

2.3.2 Códecs y contenedores

Códec es la abreviatura de Codificador-Decodificador. Describe una especificación desarrollada en software, hardware o una combinación de ambos, capaz de transformar un archivo con un flujo de datos o una señal. Los códecs pueden codificar el flujo o la señal y recuperarlo o descifrarlo del mismo modo para la reproducción o la manipulación en un formato más apropiado para estas operaciones. Los códecs son usados en videoconferencias y emisiones de medios de comunicación *[14]*.

La mayor parte de códecs provoca pérdidas de información para conseguir un tamaño lo más pequeño posible del archivo destino. Existen códecs sin pérdidas (lossless), pero en la mayor parte de aplicaciones prácticas, para un aumento casi imperceptible de la calidad no merece la pena un aumento considerable del tamaño de los datos. La excepción es si los datos sufrirán otros tratamientos en el futuro. En este caso, una codificación repetida con pérdidas a la larga dañaría demasiado la calidad.

A continuación se muestra en la *Tabla 2* algunos ejemplos de los diferentes tipos de códecs y una breve explicación sobre su funcionamiento.

CODEC	USO Y CARACTERÍSTICAS
H.261	Se utiliza principalmente en productos de videoconferencia y videotelefonía. Se incluyen por ejemplo, los conceptos establecidos así como la representación de color YCbCr, el formato de muestreo 4:2:0, 8-bits de precisión de la muestra, 16 × 16 macrobloques, compensación de movimiento , DCT de 8 × 8 bloques, exploración en zigzag, cuantificación escalar.
MPEG-1 Parte 2	Se usa para CDs de vídeo, y en ocasiones para el vídeo en línea. MPEG-1 sólo es compatible con exploración progresiva de vídeo.
MPEG-2 Parte 2	Se utiliza en DVD, SVCD, y en radiodifusión de vídeo digital y distribución de sistemas de cable. En términos de diseño técnico, la mejora más significativa en formato MPEG-2 con respecto a MPEG-1 fue la adición de soporte para entrelazado de vídeo.
H.263	Se utiliza principalmente para videoconferencia y vídeo por internet. H.263 representa un importante paso adelante en la capacidad de compresión estándar para la exploración progresiva de vídeo.
MPEG-4 Parte 2	MPEG estándar que se puede utilizar para Internet, de difusión, y en medios de almacenamiento. Se ofrece una mejor relación de calidad respecto a MPEG-2 y la primera versión de H.263. Al igual que MPEG-2, soporta escaneo progresivo y entrelazado de vídeo.
DivX , Xvid , FFmpeg y 3ivx	Distintas implementaciones de MPEG-4 Parte 2.
MPEG-4 Parte 10	Es el nuevo estándar adoptado por la UIT-T y MPEG, el cual está ganando rápidamente la adopción en una amplia variedad de aplicaciones. Contiene una serie de avances significativos en la capacidad de compresión, y recientemente ha sido adoptado en una serie de productos de la empresa, incluyendo por ejemplo el Xbox 360, PlayStation Portable, iPod, iPhone, Mac OS X v10.4, así como de HD DVD/Blu-ray Disc.
WMV (Windows Media Vídeo)	Familia de codecs de vídeo de Microsoft incluyendo WMV 7, WMV 8 y WMV 9. Se puede utilizar desde vídeo de baja resolución para el acceso telefónico a usuarios de Internet hasta HDTV.

Tabla 2: Tipos de códecs [15]

Un contenedor es un tipo de formato de archivo que almacena información de vídeo, audio, subtítulos, capítulos, meta-datos e información de sincronización necesaria para la reproducción de los distintos flujos de datos siguiendo un formato preestablecido en su especificación.

Cuando un archivo debe ser reproducido, en primer lugar actúa un divisor (*splitter*), el cual conoce el patrón del contenedor, y "separa" (demultiplexa) las pistas de audio y vídeo. Una vez separadas, cada una de ellas es interpretada por el decodificador y se reproduce.

En aquellos contenedores con más de una pista, es el usuario el que selecciona la que se va a reproducir. Es pues imprescindible que el reproductor cuente con los decodificadores necesarios para reproducir tanto el vídeo como el audio, ya que de lo contrario la información no puede ser interpretada de forma correcta.

A continuación se muestra en la *Tabla 3* algunos ejemplos de los diferentes tipos de contenedores y una breve explicación.

CONTENEDORES	USO
CONTENEDORES EXCLUSIVOS DE AUDIO	
AIFF	Formato de archivo IFF, ampliamente utilizado en la plataforma Mac OS.
WAV	Formato de archivo RIFF, ampliamente utilizado en la plataforma Windows
CONTENEDORES EXCLUSIVOS DE IMÁGENES	
TIFF	Es un formato de archivo contenedor de imágenes fijas y los metadatos asociados.
CONTENEDORES MULTIMEDIA	
3GP	Formato utilizado por gran variedad de teléfonos móviles.
ASF	Formato contenedor estándar de Microsoft WMA y WMV.
AVI	Formato contenedor estándar de Microsoft Windows, también se basa en RIFF.
MOV	Formato contenedor estándar para vídeo de QuickTime de Apple Inc.
MPEG-PS	Formato contenedor estándar para MPEG-1 y MPEG-2.
MPEG-TS	Formato contenedor estándar para la radiodifusión digital, normalmente contiene múltiples flujos de vídeo y audio.
MP4	Formato contenedor estándar de audio y vídeo para el formato MPEG-4.
RealMedia	Formato contenedor estándar de RealVideo y RealAudio.

Tabla 3: Tipos de Contenedores [15]

2.3.3 MPEG-2

MPEG-2 (Moving Pictures Experts Group 2), es la designación para un grupo de estándares de codificación de audio y vídeo acordado por *MPEG*, y publicados como estándar *ISO (International Organization for Standardization)* 13818. *MPEG-2* es por lo general usado para codificar audio y vídeo para señales de transmisión, que incluyen *TDT (Televisión Digital Terrestre)*, por satélite o cable. *MPEG-2*, con algunas modificaciones, es también el formato de codificación usado por los discos *SVCD (Super Vídeo Compact Disc)* y *DVD (Digital Vídeo Disc)* comerciales de películas *[16]*.

MPEG-2 es similar a *MPEG-1*, pero también proporciona soporte para vídeo entrelazado (el formato utilizado por las televisiones). *MPEG-2* vídeo no está optimizado para bajas tasas de *bits* (menores que 1 *Mbit/s*), pero supera en desempeño a *MPEG-1* a 3 *Mbit/s* y superiores.

MPEG-2 introduce y define flujos de transporte, los cuales son diseñados para transportar vídeo y audio digital a través de medios impredecibles e inestables, y son utilizados en transmisiones televisivas. Con algunas mejoras, *MPEG-2* es también el estándar actual de las transmisiones en *HDTV (High Definition Television)*. Un decodificador que cumple con el estándar *MPEG-2* deberá ser capaz de reproducir *MPEG-1*.

MPEG-2 audio, definido en la Parte 3 del estándar, mejora a *MPEG-1* audio al alojar la codificación de programas de audio con más de dos canales. La parte 3 del estándar admite que sea compatible hacia atrás, permitiendo que decodificadores *MPEG-1* audio puedan decodificar la componente estéreo de los dos canales maestros, o en una manera no compatible hacia atrás, la cual permite a los codificadores hacer un mejor uso del ancho de banda disponible. *MPEG-2* soporta varios formatos de audio, incluyendo *MPEG-2 AAC (Advanced Audio Coding)*.

Dentro del estándar de *MPEG-2* se encuentran dos formatos contenedores: *MPEG-TS (Transport Stream)* y *MPEG-PS (Program Stream)*. Se trata de estándares basados en cómo el vídeo y el audio es transportado, y no cómo es codificado. La calidad del vídeo no se ve modificada por el tipo de contenedor que lo contenga.

MPEG-PS

Un flujo de programa es un conjunto de paquetes bien formados *PES (Packetized Elementary Stream)* referenciados con la misma base de tiempo. Un *PS* contiene únicamente un canal de contenidos. Este estándar está preparado para el envío de material audiovisual en entornos libre de errores y con una capacidad de procesamiento de datos sencilla. Además, es mayoritariamente utilizado por aplicaciones de almacenamiento *[17]*.

MPEG-TS

En un flujo de transporte cada paquete *PES* es dividido en paquetes de tamaño fijo con la posibilidad de combinar uno o más programas con independientes marcas de tiempo. Este tipo de multiplexación está preparada para escenarios donde hay una mayor probabilidad de pérdida de paquetes o corrupción de los mismos debido al ruido. También está preparado para el envío de diferentes flujos de un programa simultáneamente *[17]*.

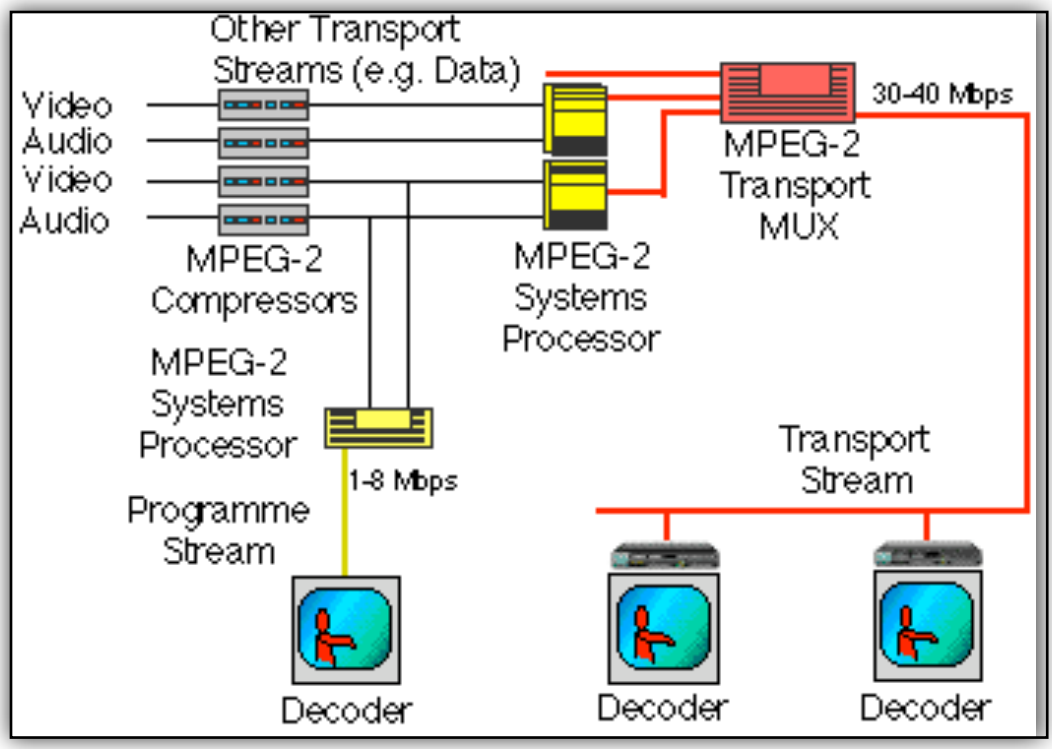


Ilustración 7: Esquema de MPEG Transport Stream [17]

Los paquetes *MPEG* generados, son introducidos en la capa de Transporte, la cual no posee un mecanismo de fiabilidad de entrega de los paquetes. En la *Ilustración 8* se muestra cómo es multiplexada la información de audio y vídeo en un paquete de transporte *MPEG2*:

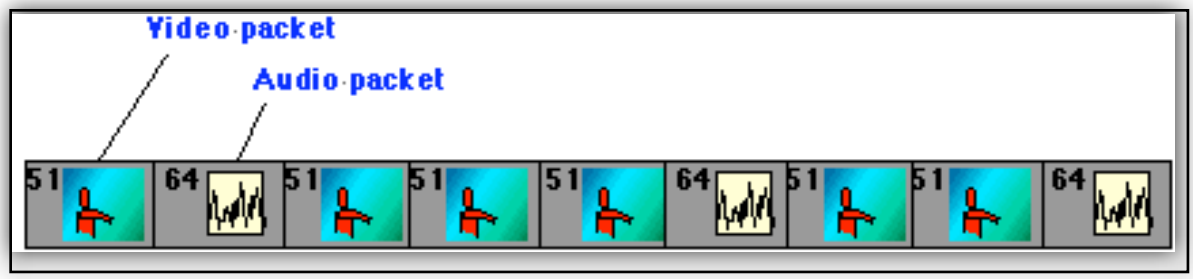


Ilustración 8: Paquete MPEG2 Transport Stream [17]

Un paquete de transporte consiste en paquetes de una secuencia fija de 188 *bytes*, de los cuales 184 son de *payload* (carga de datos) y 4 de cabecera. Cada tipo de paquete es identificado con un *PID (Packet Identifier)*, el cual indica el flujo del que proviene la información.

Normalmente, existe un mayor número de paquetes de vídeo que de audio y estos no tiene por qué estar equiespaciados. En el destino se requiere un proceso para la sincronización de los flujos.

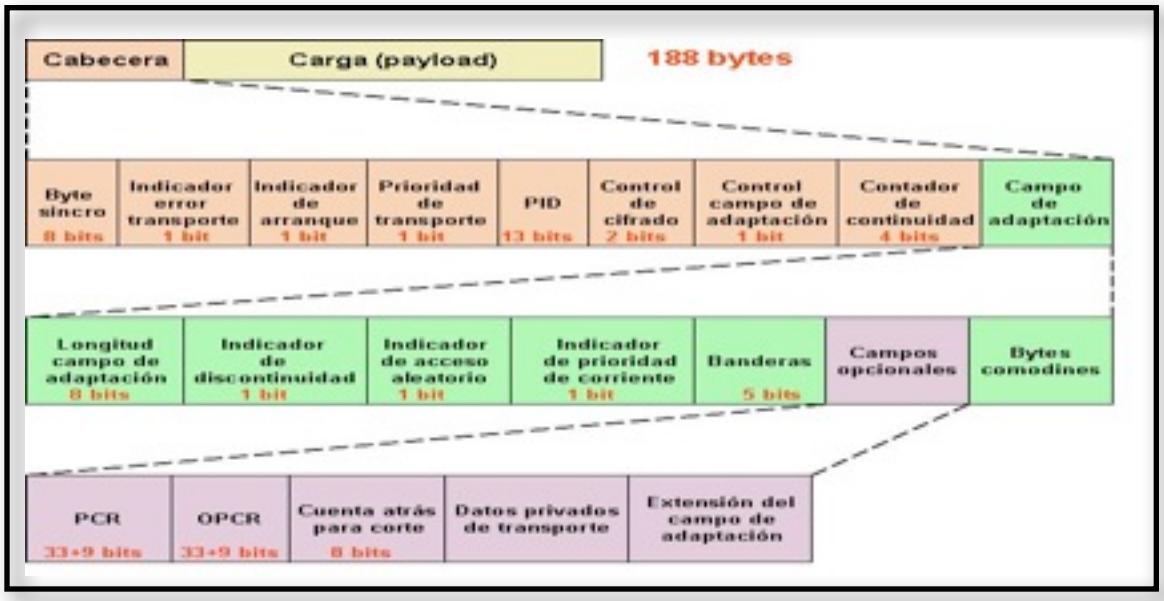


Ilustración 9: Esquema de un paquete TS [18]

A continuación, como se muestra en la *Ilustración 9*, se hace una breve descripción de los datos que aparecen en la cabecera de un paquete *TS [18]*:

- **Byte sincro**: sirve para que el decodificador pueda sincronizarse correctamente con los datos entrantes. Tiene el valor 0x47 y delimita el inicio de un paquete *TS*. Hace falta mencionar que, al contrario de los paquetes *PES*, este valor de sincronización puede darse en cualquiera de los 187 bytes restantes.

- **Indicador de error de transporte**: este bit se pone activo cuando se detecta un error en la transmisión.

- **Indicador de arranque**: indica si en la cabecera del *payload* hay un *PES*.

- **PID**: como ya se ha mencionado, los paquetes de *TS* pueden traer información de programas diferentes, además de datos para la reconstrucción de la información. Aparece un campo de 13 bits que se denomina *PID* que permite la distinción de paquetes de diferentes flujos elementales. De los 2^13 valores posibles, hay 17 reservados para funciones especiales. Esto permite 8175 valores que son asignables a todos los otros flujos elementales que forman el *TS*. El multiplexor tiene que garantizar que cada flujo elemental tenga un único *PID*. La normativa *MPEG* no especifica qué valores de *PID* se tienen que dar a los flujos elementales (a excepción de los 17 mencionados).

- **Control de cifrado**: indica si hay o no datos cifrados en el *payload*.

- **Control campo de adaptación**: indica si la cabecera tiene campo de adaptación.

- **Control de carga**: indica si hay o no datos de *payload* (no sale en el gráfico), se suele tomar el Control campo de adaptación como 2 bits y según sea 10, 01, 11 nos indica si hay de adaptación, de carga o de ambos.

- **Contador de continuidad**: el codificador lo incrementa en 1 cada vez que envía un paquete de la misma fuente. Esto permite que el decodificador sea capaz de deducir si ha habido una pérdida (o ganancia incluso) de un paquete de transporte y evitar errores que no se podrían deducir de otra manera.

Los campos más destacables dentro del Campo de Adaptación de una cabecera son los siguientes:

- **Longitud del campo de adaptación**: indica la longitud de la cabecera extra.

- **Indicador de discontinuidad**: está en el *PCR (Program Clock Reference)* y en el contador de continuidad. Se utiliza para evitar pérdidas de información producidas por un salto en el codificador.

- **PCR**: en *MPEG-TS*, el *PCR* es una información de sincronización del reloj de 27 *MHz (Megaherzio)* del receptor necesaria para la decodificación del vídeo, audio y datos. Se incluye periódicamente en los paquetes de transporte. El receptor necesita esta información – a una cadencia de unas 10 veces por segundo para hacer funcionar el bucle de fase de su oscilador local. Síncrono y en fase con el reloj *PCR* de 27 *MHz* se dispone de otro reloj de 90 *KHz (Kiloherzio)* que se necesita en el sistema para sincronizar otras funciones.

- **Bytes comodines**: son bytes de relleno para conseguir una trama de 188 *bytes* de información en el supuesto de que no hubiera información suficiente para llenar el paquete.

- **Cuenta atrás para corte**: indicador que permite una conmutación de paquetes limpia entre un *TS* y otro *TS*.

2.4 Resumen

En esta capítulo se han podido conocer y estudiar las diferentes tecnologías que van a ser utilizadas para la elaboración del proyecto. En un primer momento se ha estudiado el concepto de *middleware* y sus diferentes tipos, lo que nos ha aportado un visión más certera sobre este tipo de tecnología.

Posteriormente, se ha indagado más en los elementos que van a ser utilizados más adelante, como son DDS y el paradigma de publicación-suscripción que utiliza este tipo de sistema, base de la aplicación final que se quiere realizar. Además, gracias al estudio sobre el multimedia, y en concreto sobre el concepto de *streaming* como tipo de difusión de contenidos, se conoce la manera de la cual se quiere conseguir que la aplicación funcione.

Por todo ello, el estudio del arte previo, nos ha dado las bases para poder continuar con el desarrollo del proyecto fin de carrera y profundizar en el estudio sobre las herramientas y tecnologías a emplear.

Capítulo 3: Estudio de las tecnologías y herramientas empleadas

"El verdadero progreso es el que pone la tecnología al alcance de todos".

Henry Ford *(1863-1947) Fundador de la compañía Ford Motor Company*

En este Capítulo se realizan las primeras aproximaciones a las tecnologías que se van a utilizar en el desarrollo. Para ello, en primer lugar, se realiza un estudio del sistema de distribución OpenSplice, su configuración, arquitectura y servicios; así como la modificación, interacción y pruebas de las aplicaciones de ejemplo que trae consigo la instalación. En segundo lugar, se realiza una aproximación y estudio de las herramientas FFmpeg y SDL, utilizadas en el posterior desarrollo, así como a aplicaciones de ejemplo que ayudan a comprender mejor el uso de las mismas. Además, se explica la realización de una aplicación inicial sencilla que utiliza ambas tecnologías.

3.1 OpenSplice

La disponibilidad en tiempo real de la información es de suma importancia en los actuales sistemas de red. La información generada a partir de fuentes múltiples debe ser distribuida y puesta a disposición de las partes interesadas teniendo en cuenta la calidad de servicio entre la información generada por los productores y la información recibida por los consumidores.

Especialmente, en sistemas de tiempo real y misiones críticas, obtener los datos correctos en el momento adecuado en el lugar correcto no es una tarea trivial y hasta hace poco, no existían normas ni productos que abordaran este reto en una solución integrada. La *OMG* reconoció la necesidad de crear un sistema de distribución de datos y miembros de la organización con amplia experiencia tanto a nivel tecnológico (creación de redes y gestión de la información), así como a nivel de requisitos de usuario, crean las especificaciones para *DDS*. Estas son un conjunto coherente de perfiles cuyo objetivo es la información en tiempo real, ya sea a pequeña escala, sistemas integrados de control, o la gestión empresarial a gran escala *[19]*:

- **Perfil mínimo**: este perfil mínimo utiliza el conocido paradigma de publicación/suscripciónuscripción para poner en práctica la difusión de información altamente eficiente entre varios editores y suscriptores que comparten un mismo interés llamado "Topic". Este perfil incluye el marco de calidad de servicio.

- **Perfil Propietario**: este perfil permite la réplica por parte de los publicadores sobre la misma información a publicar, por lo que la información con "mayor fuerza" será a la que podrán suscribirse los interesados.

- **Perfil de suscripción de contenido**: este perfil posee un filtro a la hora de suscribirse al contenido de una información específica mediante consultas dinámicas a "Topics" mediante *SQL (Structured Query Language)* para el acceso en tiempo real de la información.

- **Perfil Persistente**: este perfil ofrece tolerancia a fallos y transparencia en la disponibilidad de datos no "volátiles" mediante el almacenado de los mismos en todo el sistema de distribución.

- **Perfil DLRL**: este perfil amplia el modelo centrado de datos *DCPS (Data-Centric Publish-Subscribe)* con una visión orientada a objetos en un conjunto de temas relacionados con lo que proporciona las características típicas de la orientación a objetos, como son la navegación, la herencia y el uso de tipos de valor.

Existen varias implementaciones para este estándar que trata las comunicaciones publicador/subscriptor:

- Real Time Innovations, Inc (RTI Data Distribution Service)
- PrismTech (OpenSplice DDS)
- Object Computing Inc (OpenDDS)
- MilSOFT (MilSOFT DDS)
- Twin Oaks Computing, Inc. (CoreDX)
- Gallium Visual System (InterCOM DDS)
- Icoup – technology and management consulting (MicroDDS)

OpenSplice es la implementación del estándar de la empresa *PrimTech*. El 13 de enero de 2009 la empresa publica *OpenSplice DDS Community Edition*, versión libre, bajo la licencia *GNU LGPL*. Además de la *Community Edition* existen otras tres implementaciones extendidas privadas las ediciones *Compact*, *Profesional* y *Enterprise*.

- **Community Edition**

  - Versión gratis sin costes adicionales
  - Código libre bajo la licencia *LGPL*
  - Implementación de *DDS* completa
  - Interoperabilidad con los protocolos *DDSI/RTPS*
  - Distribución de datos sobre redes en tiempo realizadas
  - Compatibilidad con *CORBA*
  - Posee más características que otras implementaciones *DDS*

- **Compact Edition**

  Todas las características que existen en la versión Community, más:

  - Posibilidad de utilizar *Eclipse* para la realización de los proyectos
  - Herramientas para realizar pruebas de las aplicaciones *DDS*
  - Subscripciones comerciales sobre *PrismTech*

- **Professional Edition**

  Todas las características que existen en la versión *Compact*, más:

  - Implementación completa del estándar *OMG DDS-DLRL* nativo en C++ y Java
  - Conectores web

- **Enterprise Edition**

  Todas las características que existen en la versión *Professional*, más:

  - Extensión de seguridad sobre *DDS*
  - Conexiones a base de datos (*MySQL, Oracle*, etc)

OpenSplice posee una arquitectura que utiliza una memoria compartida que interconecta tanto a las aplicaciones que residen dentro de un nodo de computación como a un conjunto de servicios, eso produce la escalabilidad, flexibilidad del sistema. Estos servicios proporcionan funcionalidades tales como la creación de entornos (provistos de calidad de servicio para datos en tiempo real basada en multidifusión confiable de múltiples canales) y durabilidad (para el almacenamiento tolerante a fallos para los datos).

OpenSplice utiliza una arquitectura de memoria compartida donde los datos se encuentran física y únicamente una vez en cualquier máquina, además de proporcionar a cada suscriptor una vista privada de esos datos. Esta arquitectura de memoria compartida da como resultado una mejor escalabilidad y rendimiento en comparación con las implementaciones donde cada lector/ escritor son comunicadores finales, cada uno con su propio almacenamiento, y donde los propios datos tienen que ser movidos incluso dentro del mismo nodo físico. En la *Ilustración 10* se puede observar la arquitectura de conexión de OpenSplice.

OpenSplice se puede configurar manualmente mediante la modificación de archivos *XML*, especificando los servicios que se utilizarán, así como la configuración de dichos servicios dentro del dominio de la aplicación (parámetros de red, niveles de durabilidad de los datos, etc).

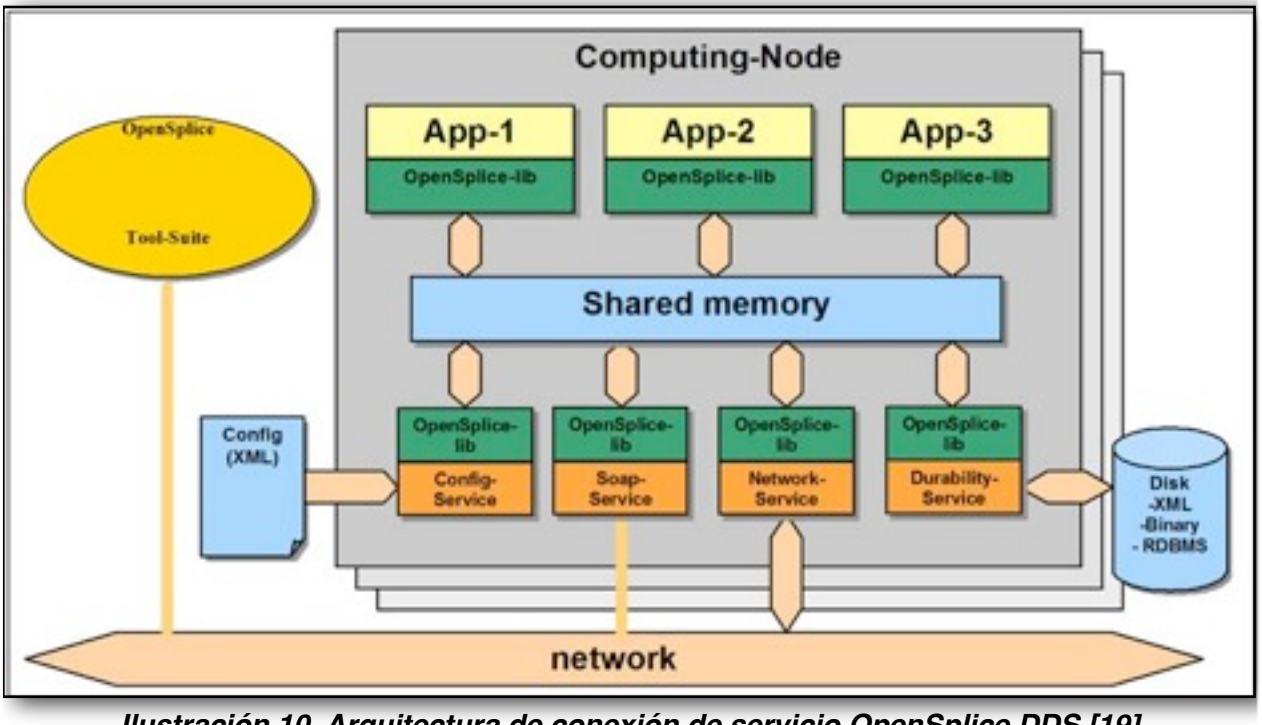


Ilustración 10. Arquitectura de conexión de servicio OpenSplice DDS [19]

OpenSplice provee una infraestructura y una capa *middleware* para sistemas distribuidos en tiempo real basadas en las especificaciones de la *OMG* de manera que puede ser añadido a cualquier aplicación de una manera sencilla. Las ventajas de utilizar *OpenSplice* son:

- Estar basado en un paradigma conceptualmente sencillo como es publicación-suscripción.
- Estar diseñado de modo que la sobrecarga del sistema sea mínima
- Ser escalable dinámicamente
- Soportar comunicaciones de uno a uno, de uno a muchos o de muchos a muchos
- Facilitar al desarrollador la tarea de desarrollo de aplicaciones para distribución de datos.

La documentación consultada ofrece las bases para un conocimiento superficial del sistema DDS. Para una mayor comprensión, se debe indagar en las configuraciones y aplicaciones de ejemplo que posee, gracias a las cuales se consigue mejorar el entendimiento hacia este tipo de *middleware*.

3.2 Primeras aplicaciones OpenSplice

Para comenzar a utilizar las dos aplicaciones de ejemplo que contiene la instalación de OpenSplice, en primer lugar, se debe conocer la configuración por defecto que trae. Para ello se puede conocer y modificar mediante la consulta al fichero *ospl.xml* que se encuentra en los Apéndices.

En este fichero se puede observar cómo está definida la memoria del sistema, es decir, la cantidad de datos que puede mantener que en este caso son 10 Mb, y la definición de los dos servicios fundamentales para el correcto funcionamiento del entorno. Por un lado se encuentra el servicio *Networking*, que es el que permite que el envío de los datos se realice por la red y por defecto está estipulado que el envío sea mediante *broadcast*. Por otro lado el servicio *Durability* es el que permite al sistema mantener los datos dentro del entorno, este servicio se puede configurar posteriormente con los parámetros de QoS.

Una vez conocida la configuración, únicamente queda arrancar el sistema de OpenSplice mediante el siguiente comando:

```
$ ospl start
```

Ping-Pong

La primera aplicación se llama *PingPong*. Consiste en un programa que realiza una evaluación comparativa entre el envío de un “topic” por parte de la aplicación Ping y el reenvío del mismo por parte de la aplicación Pong.

Este ejemplo mide, y muestra en una tabla, los tiempos de ida y vuelta de los "topics", dando una primera impresión sobre algunas de las características de rendimiento de *OpenSplice*. Los parámetros introducidos por la línea de comandos controlan variables tales como la carga útil de los "topics" (el tipo de dato que va a ser escrito dentro del tópico) y las particiones en las que se escribe y lee la información.

Las pruebas que se han realizado con esta aplicación han sido la medida de los tiempos en función del tipo de datos que se está enviando:

- Tiempo de escritura (*Write-access time*): se mide el tiempo que tarda en realizar la escritura de los datos en el dominio de distribución.
- Tiempo de lectura (*Read-access time*): se mide el tiempo que se tarda en realizar la lectura de los datos que se encuentran en el dominio de distribución
- Tiempo entre escritura-lectura (*Roundtrip time*): se mide el tiempo que transcurre entre que se han escrito los datos y se leen los mismos.

Para ello se calculan los valores mínimos (*min*), máximos (*max*) y medios (*mean*) en microsegundos. Las medidas se realizan veinte veces en cada ejecución y los datos son transmitidos en cien ocasiones entre la aplicación Ping y la aplicación Pong. Únicamente varían los datos que van a ser transmitidos.

Medida 1

En esta primera medida se realiza el cálculo sobre el envío de datos vacíos, mostrando los resultados en la *Tabla 4*:

	Roundtrip time [us]				Write-access time [us]				Read-access time [us]			
Block	Count	mean	min	max	Count	mean	min	max	Count	mean	min	max
0	100	291	0	21055	100	16	0	115	100	7	0	33
1	100	77	0	879	100	12	0	46	100	7	0	17
2	100	285	0	12965	100	13	0	74	100	7	0	20
3	100	89	15	882	100	21	1	757	100	7	1	24
4	100	6975	0	685538	100	12	0	38	100	8	0	33
5	100	102	59	1370	100	10	8	40	100	14	11	45
6	100	91	70	1410	100	9	8	25	100	13	11	99
7	100	80	1	655	100	9	0	41	100	16	0	372
8	100	76	57	461	100	8	0	19	100	12	0	23
9	100	77	70	614	100	9	8	22	100	12	11	18
10	100	78	13	786	100	16	0	712	100	12	0	29
11	100	73	69	283	100	8	8	18	100	12	11	14
12	100	77	69	325	100	8	8	18	100	12	11	21
13	100	78	59	862	100	9	3	19	100	11	4	15
14	100	76	0	547	100	14	0	489	100	9	0	14
15	100	77	0	681	100	10	0	49	100	7	0	30
16	100	68	7	517	100	12	0	294	100	6	0	14
17	100	66	56	838	100	9	8	23	100	6	5	14
18	100	82	1	677	100	14	0	560	100	11	0	53
19	100	74	65	366	100	8	7	36	100	11	11	20

Tabla 4. Medidas Ping-Pong datos vacíos

Como se puede observar, tras las veinte ejecuciones de la aplicación, el tiempo de escritura es mayor que el tiempo de lectura en todos los casos, con un valor máximo de escritura de 757 microsegundos y mínimo de 0. El tiempo de lectura es prácticamente 0 en casi todos los casos con un tiempo máximo de 372. Por otro lado, los valores resultantes entre el tiempo de escritura y lectura son mucho mas elevados con respecto a los otros. Esto puede ser debido a que el procesador haya estado atendiendo otros procesos mientras se escribía y se leían los datos.

Medida 2

En la segunda medida se realiza el cálculo sobre el envío de una palabra de texto de tamaño variable, mostrando los resultados en la *Tabla 5*:

#	Roundtrip time [us]				Write-access time [us]				Read-access time [us]			
# Block	Count	mean	min	max	Count	mean	min	max	Count	mean	min	max
0	100	157	64	3899	102	51	7	3718	100	18	10	91
1	100	74	0	297	100	10	0	89	100	13	0	74
2	100	1178	63	110538	100	9	7	39	100	12	10	45
3	100	77	63	495	100	9	7	36	100	12	10	22
4	100	197	63	7966	100	10	2	53	100	14	10	67
5	100	317	46	22340	100	12	0	54	100	238	0	22282
6	100	72	50	311	100	9	7	41	100	12	0	49
7	100	203	24	12935	100	12	0	72	100	139	0	12878
8	100	90	30	422	100	16	8	90	100	12	0	368
9	100	76	0	516	100	14	0	80	100	7	0	28
10	100	245	55	18028	100	11	9	90	100	7	5	35
11	100	302	55	23940	100	10	9	27	100	6	5	25
12	100	168	6	7640	100	35	0	628	100	8	0	25
13	100	65	55	148	100	11	9	61	100	7	5	74
14	100	1111	0	102816	100	962	0	94484	100	7	0	18
15	100	87	7	272	100	64	7	239	100	8	0	70
16	100	7006	0	692253	100	6986	0	692206	100	7	0	19
17	100	1080	61	99842	100	1030	7	99720	100	11	5	49
18	100	72	63	364	100	9	7	77	100	13	10	111
19	100	431	63	36532	100	9	7	70	100	11	10	21

Tabla 5. Medidas Ping-Pong palabra de texto variable

En esta prueba, se obtienen unos valores superiores a la prueba anterior debido al tiempo que se tarda tanto en escribir como en leer la palabra que se está enviando. En la escritura, existen casos en el que se llega a multiplicar por más de 200 el tiempo máximo, produciéndose el mismo efecto en la lectura de los datos. El resultado entre el tiempo de lectura y escritura, a su vez, se ve ampliado con una gran diferencia de valores entre los valores máximos, aunque los valores medios y mínimos se mantengan más constantes.

Medida 3

En la tercera y última medida, se realiza el cálculo sobre el envío de una secuencia de datos de tamaño 100, mostrando los resultados en la *Tabla 6*:

#	Roundtrip time [us]				Write-access time [us]				Read-access time [us]			
# Block	Count	mean	min	max	Count	mean	min	max	Count	mean	min	max
0	100	123	62	884	100	91	46	352	100	10	5	46
1	100	70	62	331	100	52	45	300	100	6	5	39
2	100	97	62	3034	100	50	45	236	100	6	5	14
3	100	180	50	7634	100	159	30	7579	100	7	0	78
4	100	81	63	553	100	61	46	464	100	7	5	30
5	100	111	0	2334	100	90	0	2302	100	7	0	33
6	100	314	1	24678	100	296	1	24625	100	6	0	23
7	100	70	27	209	100	12	1	62	100	7	0	13
8	100	61	56	325	100	12	9	265	100	6	5	8
9	100	235	0	14298	100	66	0	224	100	8	0	43
10	100	370	63	30051	100	51	46	197	100	306	5	29989
11	100	69	1	219	100	52	1	184	100	6	0	15
12	100	361	25	26224	100	298	9	26057	100	8	0	33
13	100	76	6	166	100	15	0	48	100	7	0	12
14	100	82	56	750	100	44	9	734	100	7	5	24
15	100	240	0	12249	100	164	0	12064	100	8	0	25
16	100	59	57	94	100	10	9	32	100	6	6	15
17	100	92	57	2249	100	12	9	35	100	29	6	2190
18	100	71	45	588	100	12	9	95	100	7	0	25
19	100	178	16	10678	100	15	9	272	100	113	0	10626

Tabla 6. Medidas Ping-Pong secuencia 100 datos

En esta ocasión, se produce un aumento medio del tiempo de escritura de los datos, con respecto a las pruebas anteriores, debido a que se tienen que procesar más datos antes de escribirlos. Sin embargo, el tiempo de lectura se mantiene más estable, esto puede ser debido a que en este caso es un tamaño fijo de secuencia por lo que el tiempo de lectura es más estable. En el tiempo entre la lectura y la escritura, también se observa dicha estabilidad, con algunos picos de tiempo debido a la escritura de los datos.

Gracias a esta aplicación y a las pruebas realizadas, se puede comprender mejor el envío, recepción y los tiempos que transcurren entre ambos, así como las diferencias que existen entre los distintos tipos de datos que pueden ser enviados. Además, la aplicación ayuda a la posterior elección de los tipos de datos que se van a enviar en la aplicación final, así como a poder implementar la manera más rápida y eficaz de enviar dichos datos.

Chat

La segunda aplicación consiste en un ejemplo muy primitivo de Chat. Esta aplicación consta de tres archivos ejecutables independientes. Cada ejecutable del Chat se puede iniciar con configuraciones diferentes, ya que cada uno se ejecuta independientemente en su ventana (un terminal de Linux), de esta forma se evita que se mezcle la información de los diferentes ejecutables en una misma pantalla.

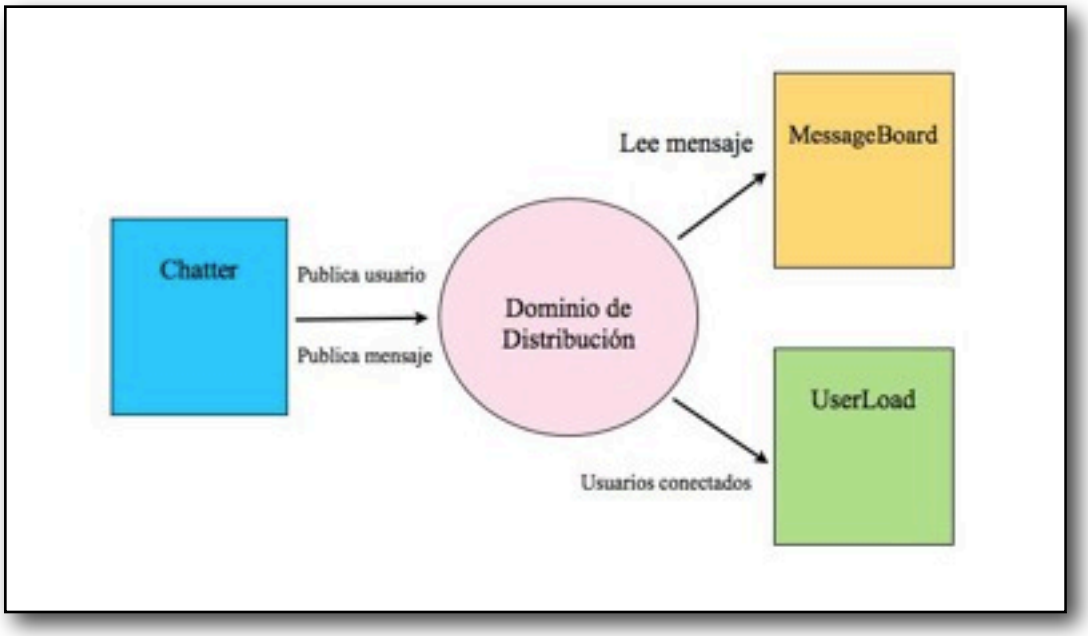


Ilustración 11. Agentes del ejemplo de Chat

Como se muestra en la *Ilustración 11*, los tres agentes que intervienen en la aplicación de Chat, son los siguientes:

- ***Chatter***: esta parte es la responsable de la publicación de la identidad de los usuarios dentro del dominio de distribución, así como el envío de los mensajes que los integrantes del Chat quieren transmitir. El usuario envía mensajes equiespaciados en el tiempo hasta un total de diez. Esta aplicación únicamente escribe datos en el sistema *DDS*. Por cada usuario que se quiera conectar al Chat se debe de arrancar una aplicación *Chatter*.

- **MessageBoard**: esta parte es la responsable de suscribirse a todos los mensajes introducidos en el sistema *DDS* y mostrarlos en el mismo orden en el que han sido recibidos. Esta aplicación únicamente lee los datos introducidos en el sistema *DDS*. Se trata de una aplicación única por Chat, en la que al arrancarla se le puede indicar un identificador de usuario para filtrar esos mensajes y que no aparezcan.

- **UserLoad**: esta parte es la responsable de mantener un control sobre los usuarios que entran y salen del entorno de distribución de Chat. Esta aplicación lee los datos introducidos en el sistema, en este caso los usuarios que se conectan y abandonan el Chat, y aunque es prescindible para las comunicaciones, permite conocer el estado del Chat en cuanto a usuarios conectados.

Las pruebas que se han realizado con esta aplicación es la interacción entre los diferentes elementos del Chat, así como el funcionamiento del mismo cuando interviene únicamente un usuario, cuando intervienen dos o cuando el *MessageBoard* se inicia posteriormente a que un usuario se haya conectado al Chat. Todo ello supervisado por la aplicación *UserLoad* para conocer en todo momento los usuarios que están interviniendo en la comunicación.

Prueba 1. Un usuario

En esta prueba, se inician en primer lugar, tanto el *MessageBoard* como el *UserLoad*, y acto seguido se inicia el *Chatter*:

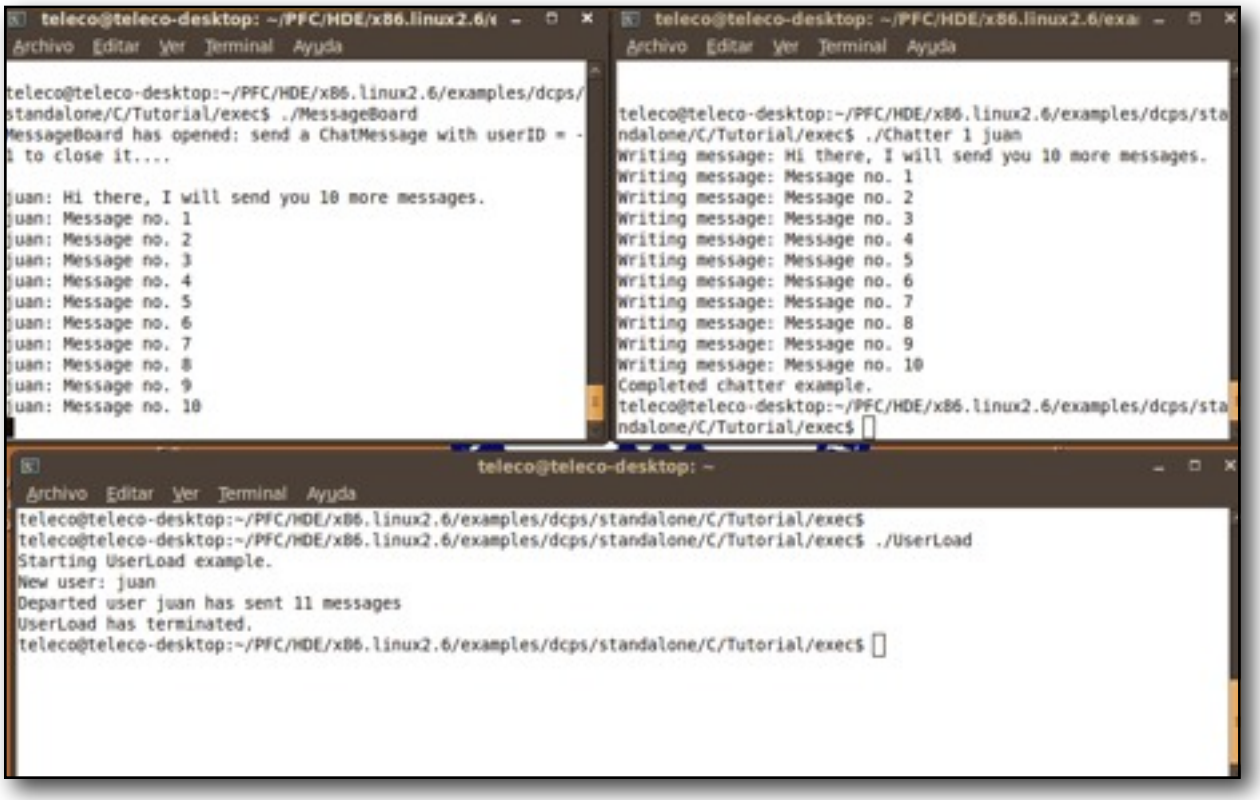


Ilustración 12. Ejemplo Chat un usuario

Como se puede observar en la *Ilustración 12*, la consola de la esquina superior izquierda se encuentra el *MessageBoard*, en la parte inferior el *UserLoad* y en la superior derecha el *Chatter*. Una vez arrancados los dos primeros, se mantienen a la espera de que un usuario se conecte y comience a enviar mensajes. Cuando el *Chatter* es iniciado el *UserLoad* obtiene el nombre del usuario conectado y lo muestra, mientras que el *MessageBoard* obtiene los mensajes de texto que está enviando el usuario. Una vez, el usuario abandona el Chat, el *UserLoad* lo notifica y muestra unas estadísticas sobre los mensajes enviados por el usuario.

Se trata de un caso óptimo, donde un usuario se conecta cuando el resto de aplicaciones ya están iniciadas, por lo que todo sucede como se espera, se reciben todos los mensajes y en el mismo orden en el que se enviaron.

Prueba 2. Dos usuarios

En esta prueba, se inician en primer lugar, tanto el *MessageBoard* como el *UserLoad*, y acto seguido se inician dos aplicaciones *Chatter* con un intervalo de dos segundos:

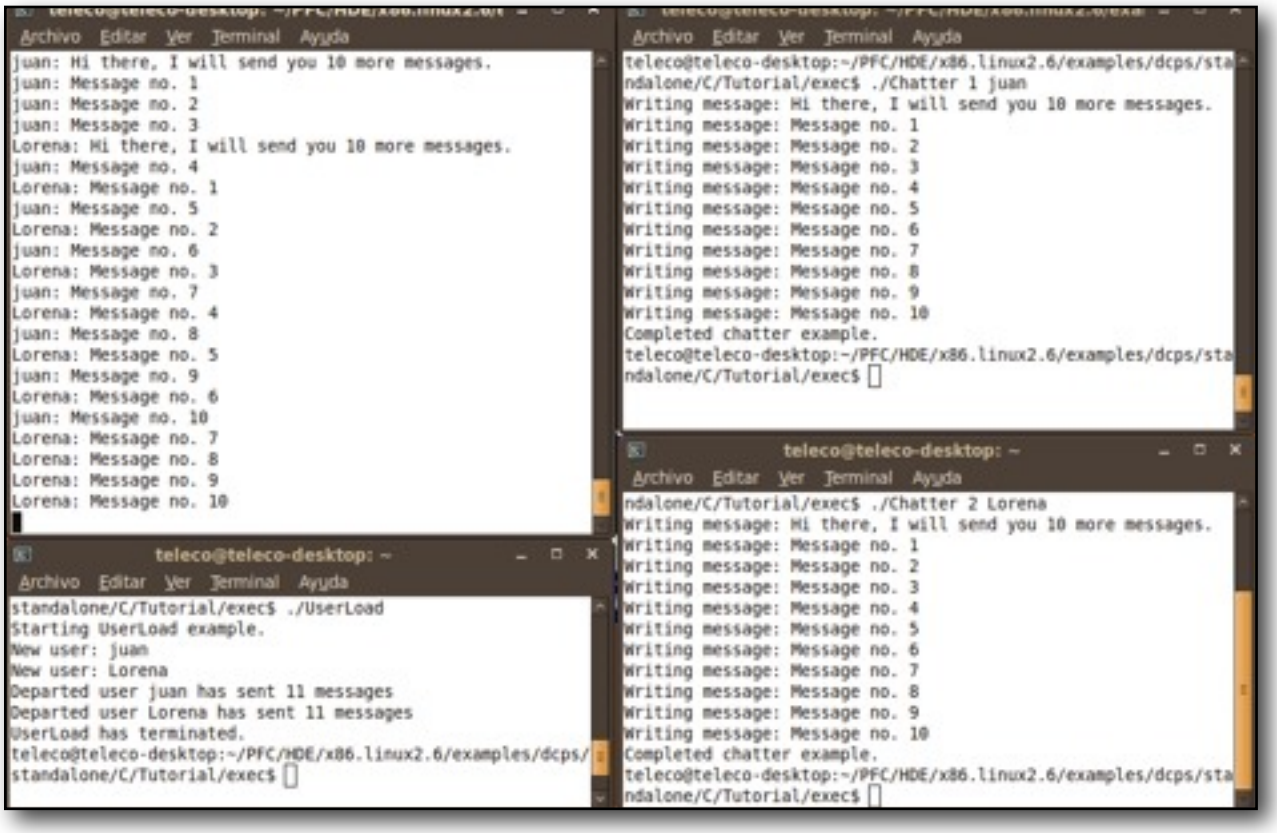


Ilustración 13. Ejemplo Chat dos usuarios

En esta prueba se tiene el *MessageBoard* en la esquina superior izquierda, el *UserLoad* en la inferior izquierda y los *Chatter* en la parte de la derecha (*Ilustración 13*). Al igual que en la prueba anterior, las dos primeras aplicaciones se han lanzado antes que los usuarios. Como se puede ver en el *UserLoad* "juan" se ha conectado antes que "Lorena" y por ello se han recibido primero los mensajes de "juan" en el *MessageBoard* hasta que "Lorena" ha iniciado la aplicación. En ese momento los mensajes empiezan a intercalarse entre uno y otro hasta que finalmente ambos abandonan el Chat.

En esta segunda prueba comprobamos que mensajes emitidos por diferentes usuarios llegan correctamente al *MessageBoard*, estos mensajes son recibidos con el mismo orden con el que fueron enviados, pero se intercalan en función del momento en el que lo están enviando, por lo que podemos observar que no existe retardo en la comunicación.

Prueba 3. Inicio tarde del *MessageBoard* y *UserLoad*

En esta última prueba, se iniciará en primer lugar el *Chatter* y posteriormente las aplicaciones *MessageBoard* y *UserLoad*:

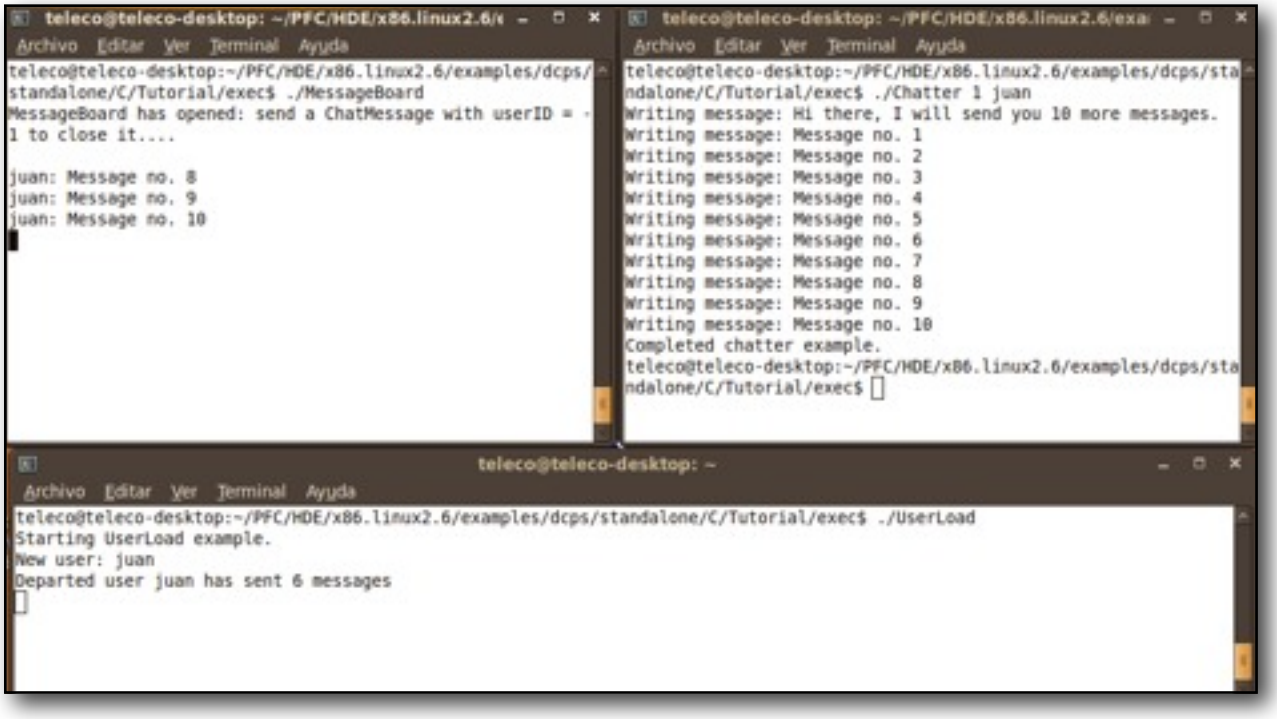


Ilustración 14. Ejemplo Chat inicio tarde

Como se puede observar en la *Ilustración 14*, la disposición de las aplicaciones es la misma a la Prueba 1. En esta ocasión se pueden sacar bastantes conclusiones de la configuración del ejemplo del Chat. Al iniciarse antes el *Chatter* que el resto de aplicaciones se ha producido una pérdida de los mensajes que han sido enviados por éste en el sistema de distribución.

Esto se ve reflejado tanto en el *UserLoad* debido a que, aunque ha registrado que ha habido un usuario conectado, únicamente ha recibido 6 mensajes desde que se ha conectado; y por parte del *MessageBoard*, la pérdida de mensajes es más apreciable, ya que únicamente ha impreso por pantalla tres de los once que obtuvo en las pruebas anteriores.

En esta prueba podemos sacar como conclusión que debido a la configuración del Chat, tanto el MessageBoard como el UserLoad, deben estar iniciados con antelación para que, una vez, un Chatter se conecte, estos consigan recibir todos los mensajes. En este ejemplo, la configuración que tiene únicamente permite mantener el último mensaje enviado en el sistema de distribución, por lo que los mensajes anteriores son sobre-escritos.

Gracias a las pruebas realizadas, se ha comprendido de una manera más completa el funcionamiento de *OpenSplice*, se ha apreciado que se trata de un sistema fiable donde los datos se obtienen de forma ordenada, así como la diversidad de situaciones que se pueden dar cuando se realizan cambios en el inicio de las aplicaciones y en la configuración de la calidad de servicio, pudiéndose producir pérdidas de paquetes cuando no se tiene la configuración adecuada.

3.3 FFmpeg y SDL

FFmpeg

FFmpeg es una colección de software libre que puede grabar, convertir y hace *streaming* de audio y vídeo. *FFmpeg* está desarrollado en *GNU (General Public License)*/Linux, pero puede ser compilado en la mayoría de los sistemas operativos, incluyendo Windows *[20][21]*.

FFmpeg es *software* libre bajo una licencia GNU 2.1+ o GNU 2+ (dependiendo de cuáles bibliotecas estén incluidas). Los desarrolladores recomiendan utilizar el último versión de *Subversion* ya que mantienen constantemente una versión estable.

El proyecto está compuesto por:

- **ffmpeg**: es una herramienta de línea de comandos para convertir un vídeo de un formato a otro. Además con este programa se puede realizar la captura y codificación en tiempo real de una tarjeta de vídeo o una cámara web.

- **ffserver**: es un servidor de *streaming* multimedia de emisiones en directo que soporta *HTTP (Hypertext Transfer Protocol)*

- **ffplay**: es un reproductor multimedia basado en *SDL (Simple DirectMedia Layer)* y las bibliotecas *FFmpeg*.

- **libavcodec**: es una biblioteca que contiene todos los códecs de *FFmpeg*. Muchos de ellos fueron desarrollados desde cero para asegurar una mayor eficiencia y un código altamente reutilizable.

- **libavformat**: es una biblioteca que contiene los multiplexadores/ demultiplexadores para los archivos contenedores multimedia.

- **libavutil**: es una biblioteca de apoyo que contiene todas las rutinas comunes en las diferentes partes de *FFmpeg*.

- **libpostproc**: es una biblioteca de funciones de postproceso de vídeo.

- **libswscale**: es la biblioteca de escalado de vídeo.

FFmpeg es un programa sin interfaz gráfica que permite convertir o transformar entre formatos multimedia, tanto de vídeo como de audio. Aunque existen otros programas, algunos sin necesidad de usar comandos, es una de las opciones con más posibilidades y es muy rápida.

Codecs Vídeo	Codecs Audio
MPEG-1	Apple Lossless
MPEG-2	Cook Codec
MPEG-4 Parte 2 (el formato utilizado por los códecs DivX y Xvid)	FLAC
H.261.	MP2
H.263	MP3
H.264/MPEG-4 AVC (únicamente la decodificación).	Shorten
WMV versión 7, 8 y 9 (únicamente la decodificación).	QDM2
Sorenson codec	RealAudio 1.0
Cinepak	RealAudio 2.0
MJPEG	Vorbis
Huffyuv	WavPack
Snow	WMA

Tabla 7. Códecs soportados por FFmpeg

En la *Tabla 7*, se muestra la variedad de códecs de audio y vídeo que son soportados, gracias a los cuales, hace que la herramienta *FFmpeg* sea muy versátil y de las más utilizadas en el entorno multimedia.

SDL

Simple DirectMedia Layer es un conjunto de librerías desarrolladas con el lenguaje C que proporcionan funciones básicas para realizar operaciones de dibujado 2D, gestión de efectos de sonido y música, y carga y gestión de imágenes *[22]*.

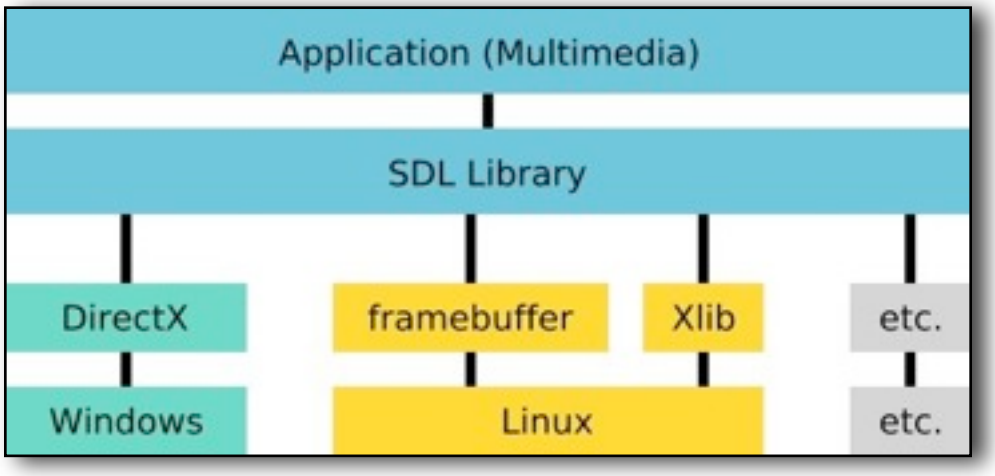


Ilustración 15. Capas de abstracción de SDL

Aunque se ha programado en C, tiene *wrappers (*conversiones*)* a otros lenguajes de programación como $C/C++$, ADA, asic, Lua, Java, Python, etc. También proporciona herramientas para el desarrollo de videojuegos y aplicaciones multimedia. Una de sus grandes virtudes es que al tratarse de una librería multiplataforma, soporta sistemas como Windows, Linux, MacOS y QNX, además de otras arquitecturas/sistemas como Dreamcast, GP32, GP2X. De ahí vienen las siglas *Simple Direct Media Layer* que más o menos alude a capa de abstracción multimedia.

Debido a la escasa documentación encontrada sobre FFmpeg y, en menor medida SDL, se encontró la necesidad de hacer un estudio exhaustivo de ambas tecnologías, así como la realización de unas primeras aplicaciones de ejemplo con el fin de conseguir conocer más a fondo el código para su posterior desarrollo.

3.4 Primeras aplicaciones FFmpeg y SDL

Para la iniciación en *FFmpeg* y *SDL* se ha utilizado una aplicación de ejemplo en la que se explica como se puede realizar la construcción de un programa que decodifica un fichero de vídeo mediante *ffmpeg* y lo muestra en una pantalla mediante *SDL*, este tutorial se encuentra en la página web de FFmpeg *[20]*.

Aunque el tutorial se divide en varias secciones, se utilizaron las dos primeras para realizar las pruebas para la comprensión del funcionamiento de la herramienta. A continuación se muestran las tutoriales seguidos.

Tutorial 1. Apertura de un vídeo y escritura de *frames* en fichero

Este tutorial consiste en la apertura de un fichero de vídeo para la lectura de las diferentes secuencias de vídeo que contiene y escribir los *frames* de cada secuencia en un fichero *PPM (Portable Pixel Map)* como se muestra en la *Ilustración 16*.

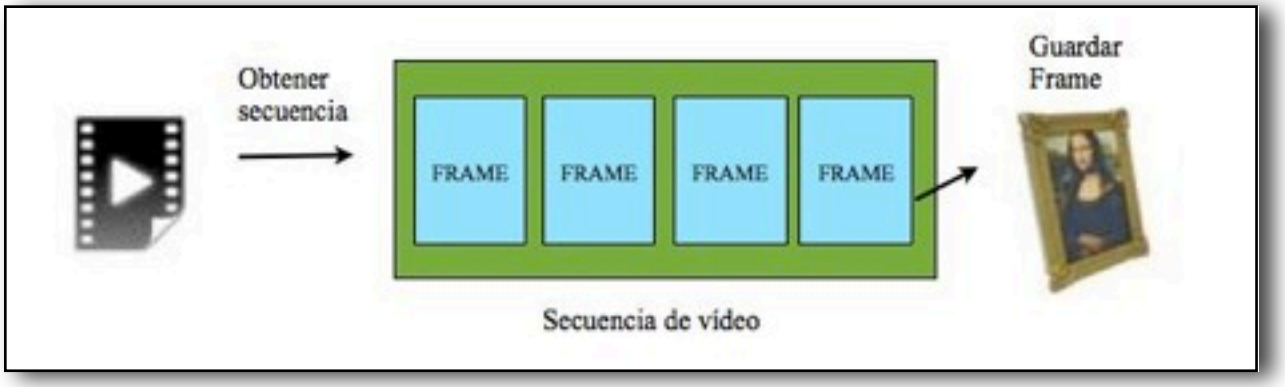


Ilustración 16. Diagrama Tutorial 1 FFmpeg

Se pueden diferenciar 6 pasos a la hora de conseguir realizar esta primera aproximación.

Paso 1. Registro de formatos

Para la iniciación del vídeo, en primer lugar, se deben registrar todos los formatos y códecs disponibles en la biblioteca de *ffmpeg* para utilizar automáticamente el correspondiente decodificador una vez abierto el archivo de vídeo, para ello se hace la llamada a la función:

```
av_register_all();
```

Paso 2. Abrir fichero

El primer paso que hay que realizar para la apertura del fichero de vídeo es llamar a la función *av_open_input_file()* la cual lee la cabecera del archivo y almacena la información necesaria en una estructura que se le ha introducido como parámetro.

Paso 3. Obtener información vídeo

Como se ha dicho, la función anterior únicamente obtiene los datos de la cabecera, por lo que para obtener la información de la secuencia de vídeo en el archivo se debe llamar a la función *av_find_stream_info()*. Esta función rellena los flujos de datos con la información adecuada tanto de vídeo como de audio.

Paso 4. Encontrar y abrir el códec

Dentro de la información que obtenemos, la cual está guardada en una estructura determinada, se encuentra la información sobre las secuencias de vídeo y el tipo de codificación que está utilizando el flujo. Para obtener el códec que está siendo utilizado se utiliza la función *avcodec_find_decoder()* y para abrir dicho códec la función *avcodec_open()*.

Paso 5. Obtención, decodificación y conversión de los *frames*

El siguiente paso es la lectura de los paquetes a través de la secuencia de vídeo, decodificar y convertir los *frames* para finalmente guardarlos en el fichero .*PPM*. Técnicamente, un paquete puede contener *frames* parciales u otros bits de datos, pero el analizador de *ffmpeg* asegura que los paquetes que recibe contienen imágenes completas o múltiples. Para la obtención de estos *frames* se utiliza la función *av_read_frame()*, posteriormente se decodifican mediante *avcodec_decode_video()* para finalmente convertir estos *frames* al formato que se desea, en este caso *RGB (Red Green Blue),* mediante la función *img_convert()*.

Paso 6.Crear fichero de salida

Por último, se crea el fichero en el que se van a guardar los datos del *frame*, así como la altura y el ancho. El archivo *PPM* final consiste en la información *RGB* del *frame* puesta de forma lineal en un fichero de texto, donde la cabecera del fichero indica el ancho y el alto de la imagen.

Ilustración 17. Fichero frame.PPM

La *Ilustración 17* muestra el *frame* guardado sobre un vídeo de prueba. La imagen de la izquierda muestra el fichero de datos que genera en el que en la primera línea muestra un identificador de la imagen, la segunda el ancho y el alto, la tercera el valor máximo de los colores[2], y el resto corresponden a los datos en bruto de la imagen que se observa a la derecha.

Con este primer tutorial se ha conseguido conocer las diferentes estructuras y funciones que posee la herramienta *FFmpeg* y que intervienen en la apertura y decodificación de los datos de un fichero de vídeo, así como el tratamiento que se puede realizar sobre los diferentes *frames* que contiene una secuencia de vídeo.

Tutorial 2. Apertura de un vídeo y mostrarlo por pantalla

Este segundo tutorial tiene un funcionamiento parecido al primero, pero en esta ocasión las imágenes que se obtienen, en lugar de salvarlas en un fichero, se van a dibujar en la pantalla gracias a la herramienta *SDL,* como se muestra en la *Ilustración 18*.

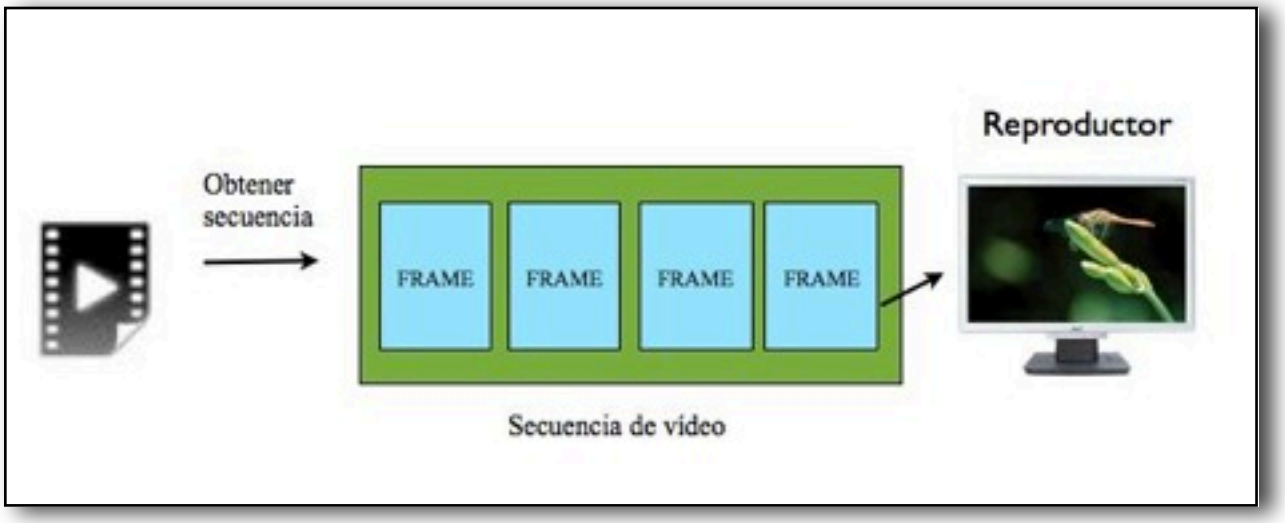


Ilustración 18. Diagrama Tutorial 2 FFmpeg

[2] El rango que de valores que pueden tomar los colores va de 0-255

Como en el *tutorial 1*, los primeros pasos que se realizan son la apertura del vídeo, la obtención de los datos de la secuencia, la búsqueda del codificador que utiliza y la decodificación de esos datos para la conseguir los *frames* que, en este tutorial, se van a visualizar por la pantalla.

A los *6 pasos* que se comentaron en el primer tutorial, habría que eliminar el paso 6 y añadir 4 pasos más para poder visualizar las imágenes en una pantalla, para llegar a un total de 9.

Paso 6. Inicializar bibliotecas de audio y vídeo

Para poder dibujar las imágenes en la pantalla, en primer lugar, se deben inicializar las bibliotecas de audio, vídeo y del temporizador de *SDL* mediante esta sentencia:

```
SDL_Init (SDL_INIT_VIDEO | SDL_INIT_AUDIO | SDL_INIT_TIMER);
```

Paso 7. Crear la pantalla

El siguiente paso es la creación de la pantalla para poder visualizar las imágenes, para ello se utiliza un componente específico de *SDL* llamado *SDL_Surface*, al cual se le establece el tamaño que debe tener mediante la anchura y altura, que viene contenida en la información del vídeo, utilizando la función *SDL_SetVideoMode()*.

Paso 8. Crear capa de imágenes

Las imágenes obtenidas son guardadas en una estructura con la información de altura y anchura de las mismas. Estas imágenes son convertidas al formato *YUV* que es el que *SDL* usa para la visualización de las mismas. Para ello se crea una capa en la que se superponen las imágenes encima de la pantalla creada utilizando la función *SDL_CreateYUVOverlay()*.

Paso 9. Reproducción de las imágenes

Por último, se crea un rectángulo con el tamaño con el que se va a visualizar el vídeo *SDL_Rect,* para finalmente reproducir las imágenes en la pantalla gracias a la función SDL_DisplayYUVOverlay. La *Ilustración 19* muestra la visualización resultante.

Ilustración 19. Captura de pantalla del vídeo reproducido [25]

En este tutorial se afianzan los conocimientos obtenidos con el anterior tutorial sobre la herramienta *FFmpeg* y el tratamiento de un fichero de vídeo. Además, se aprende a iniciar una pantalla de visualización mediante *SDL* para poder reproducir las imágenes que se obtienen del vídeo.

Aunque la documentación sobre FFmpeg, y no tanto de *SDL*, no es muy amplia fuera de estos tutoriales, gracias a ellos se ha podido aprender cómo realizar el tratamiento sobre apertura, decodificación, conversión y visualización de vídeos. Esto es muy importante para la aplicación final del proyecto ya que se consiguen las bases necesarias para poder aplicar estos conocimientos a casos similares que se van a utilizar.

3.5 Aplicación inicial (Vídeo-DDS)

Tras la realización de las pruebas y tutoriales anteriores, y el conocimiento que se ha adquirido, el siguiente cometido es conseguir una aplicación básica que reúna las dos funcionalidades:

1. Envío de datos mediante DDS
2. Transmisión del vídeo

Para ello, es necesario la comunicación entre dos agentes en el que uno de ellos envíe el vídeo y otro lo reciba y reproduzca, todo ello mediante el paradigma de publicación-suscripción que garantiza DDS. La *Ilustración 20* muestra el diagrama de la aplicación.

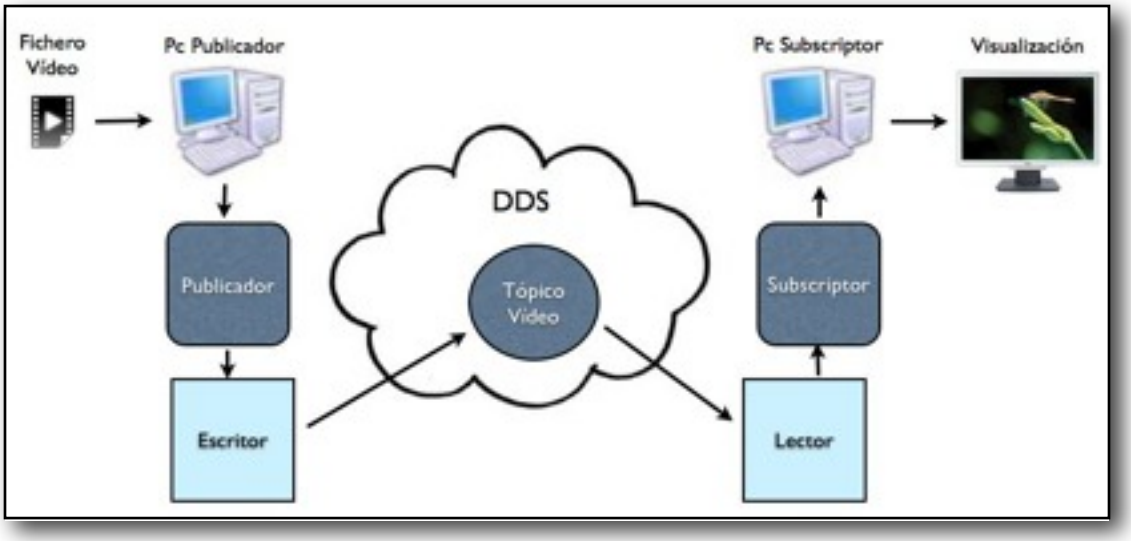


Ilustración 20. Diagrama aplicación Vídeo-DDS

Publicador de Vídeo

Este agente es el encargado de la apertura del fichero de vídeo y la posterior publicación de los datos en el sistema distribuido. Para ello a continuación se van a explicar los pasos que hay que realizar para la realización de este cometido.

Paso 1. Crear los elementos DDS

Para poder realizar la comunicación, en primer lugar, se deben de crear los elementos necesarios para la publicación y escritura de los datos dentro del sistema distribuido. Los pasos a seguir son los siguientes:

- **Crear el dominio de participación**: es el entorno en el que se van a publicar los datos y de donde los va a obtener el subscriptor. Para poder crearlo se debe llamar a la función *DDS_DomainParticipantFactory_create_participant().*

- **Registrar los datos a publicar**: los datos que van a ser publicados deben ser registrados mediante un nombre para poder ser identificados. Para poder hacer el registro se utiliza la función *TypeSupport_register_type().*

- **Configurar la QoS**: el siguiente paso es configurar la calidad de servicio que va a tener el tópico de los datos para que la comunicación sea fiable. Para ello, en primer lugar, se reserva memoria para la QoS del tópico mediante *DDS_TopicQos__alloc(),* y posteriormente se establece que tenga un tipo de envío de datos fiable mediante la siguiente la sentencia *topic_qos->reliability.kind = DDS_RELIABLE_RELIABILITY_QOS;*

- **Crear el tópico**: se crea el tópico en el que se van a publicar los datos con la configuración de calidad de servicio creada anteriormente mediante la función *DDS_DomainParticipant_create_topic()*

- **Crear el publicador**: se crea el publicador que será el encargado de gestionar los diferentes escritores de datos. Para ello se utiliza la función *DDS_DomainParticipant_create_publisher()*

- **Crear el escritor**: será el encargado de escribir los datos que se vayan obteniendo del fichero del vídeo. Para crearlo se utiliza la función *DDS_Publisher_create_datawriter()*

Paso 2. Enviar los datos del vídeo

Una vez creados los elementos DDS, se realiza la lógica para poder ir obteniendo los datos del vídeo y su posterior envío, para ello se crea una función que realiza está acción hasta que se hayan enviado todos los datos. Los pasos que realiza esta función son los siguientes:

- **Registrar el escritor y los datos**: se debe de registrar la instancia que realiza la escritura de los datos. Para ello se utiliza la siguiente función *DataWriter_register_instance().*

- **Obtener los datos del fichero**: se van leyendo los datos del fichero de vídeo y se guardan en una variable. Para la lectura se utiliza la función *fread().*

- **Escribir los datos**: publican los datos en el dominio mediante la función *DataWriter_write().*

Paso 3. Eliminar y liberar los elementos creados

Una vez finalizada la publicación del vídeo se deben de eliminar y liberar todos los recursos y elementos que se han creado en el agente, evitando así pérdidas de memoria. Este proceso es el inverso al de creación, por lo que se deben ir eliminando los elementos de forma paralelamente inversa:

- **Eliminar el registro del escritor y los datos**:

  *DataWriter*_unregister_instance()

- **Eliminar el escritor**:

  DDS_Publisher_delete_datawriter()

- **Eliminar el publicador:**

  DDS_DomainParticipant_delete_publisher()

- **Eliminar el tópico**:

  DDS_DomainParticipant_delete_topic()

- **Liberar la QoS reservada**:

  DDS_free()

- **Eliminar el dominio de participación**:

  DDS_DomainParticipantFactory_delete_participant()

Subscriptor de Vídeo

Este agente es el encargado de estar subscrito al tópico del vídeo, recibir esos datos y tratarlos para su posterior reproducción. Los pasos a realizar son similares al *paso 1* y *paso 3* del publicador, con ligeras diferencias en cuanto a los elementos que intervienen en el dominio de distribución.

A continuación se detallan las diferencias con el agente Publicador y la lógica utilizada para la reproducción.

Paso 1. Crear los elementos DDS

Los elementos que se deben de crear son iguales que en el agente Publicador, para poder tener la comunicación sobre los mismos datos, exceptuando el publicador y el escritor. En este caso, son necesarios un subscriptor y un un lector:

- **Crear el subscriptor**: se crea el subscriptor que será el encargado de gestionar los diferentes lectores de datos. Para ello se utiliza la función *DDS_DomainParticipant_create_subscriber()*

- **Crear el lector**: será el encargado de leer los datos que se encuentran publicados en el dominio de distribución dentro del tópico correspondiente. Para crearlo se utiliza la función *DDS_Subscriber_create_datareader()*

Paso 2. Recibir los datos del vídeo y reproducirlos

Los pasos a seguir para la recepción y reproducción de los datos del vídeo son básicamente los que se hicieron en el *Tutorial 1* y *Tutorial 2* de *ffmpeg* que se mostró anteriormente. La única diferencia es la forma en que se obtienen los datos (*Paso 2 del Tutorial 1*), ya que en este caso no se trata de un fichero, si no de un flujo de datos que se tienen que tratar.

Para conseguir tratar estos datos se deben de realizar 4 acciones:

- Obtener los datos del sistema distribuido mediante la función *DataReader_take()* y guardarlos en un objeto para su posterior manipulación.

- Crear de un objeto de tipo *ByteIOContext* en el que se introducirán los datos para su posterior utilización obteniendo la información sobre el vídeo.

- Implementar una función que copie los datos que se están obteniendo del tópico, por parte del lector, en el objeto anterior. El estilo de la función a implementar es el siguiente:

  *static int read_packet_stream(void *opaque, uint8_t *buf, int buf_size)*

  - *opaque:* datos leídos del tópico
  - *buf*: objeto en el que se van a copiar los datos
  - *buf_size*: número de datos que se van a copiar

- Llamar al método *init_put_byte()* el cual se encarga de realizar la llamada a la función anterior y por último, abrir la información del flujo del vídeo mediante la llamada al método *av_open_input_stream().*

Una vez realizadas estas acciones, se continúa por el *Paso 3 del Tutorial 1*, hasta completar todos y conseguir la reproducción del vídeo.

Paso 3. Eliminar y liberar los elementos creados

Al igual que en el publicador, una vez finalizada la visualización del vídeo se procede a eliminar los elementos creados. El orden es el mismo que en el caso homólogo, sustituyendo el publicador por el subscriptor y el escritor por el lector:

- **Eliminar el lector**:

  DDS_Subscriber_delete_datareader()

- **Eliminar el subscriptor**:

  DDS_DomainParticipant_delete_subscriber()

3.6 Conclusiones

Tras una primera toma de contacto con las tecnologías que se van a utilizar para el desarrollo de la aplicación final, se pueden sacar varias conclusiones:

- Los tutoriales de *DDS*, al ser muy detallados y estar bien redactados, permiten la comprensión del funcionamiento del sistema y dan una visión global del mismo, pero no muestran la gran variedad de situaciones que pueden darse en un entorno real.

- Los tutoriales seguidos para *ffmpeg* y *SDL*, marcan unas pautas mínimas para conseguir la visualización de un vídeo. Este tutorial es bastante completo, pero más allá de él, no existe mucha mas documentación que pueda ayudar a intentar realizar diferentes cosas, como es nuestro caso.

- Como se ha comentado, la documentación para *ffmpeg* es muy escasa, y debido al *API* tan grande que posee, se convierte en un trabajo muy tedioso el intentar implementar algo fuera de lo que marca el tutorial. En nuestro caso, para conocer como tratar y obtener un flujo de datos que proviene de la red y no de un fichero, hubo que indagar en los ficheros fuente (.h, .c), y comprender la lógica que realizaban para poder implementarlo.

- Gracias a la aplicación inicial implementada, combinamos las tecnologías que vamos a utilizar en el desarrollo y nos permite tener los conocimientos necesarios para el desarrollo final.

Capítulo 4: Desarrollo del sistema

"Un desarrollo inadecuado sería una tragedia".

John Opie *(1761-1807) Pintor inglés*

En este capítulo se muestran las partes de la aplicación que se ha desarrollado, justificando las decisiones tomadas durante el proceso. Además se explicarán las partes más relevantes del código de la aplicación, con el fin de proveer una visión más fina del funcionamiento de ésta. Para ello, en primer lugar, se realiza una pequeña introducción, seguida de una descripción del entorno de desarrollo. Posteriormente, se detalla la arquitectura y diseño que tiene la aplicación, así como, del desarrollo final de la misma.

4.1 Introducción

El objetivo principal del trabajo es conocer las posibilidades, facilidad de funcionamiento, e implementación de la tecnología de *middleware* ***OpenSplice***, así como la utilización de *FFmpeg* como codificador, decodificador y reproductor de contenidos multimedia.

Con este objetivo fijado desde un inicio, se establecen una serie de requisitos necesarios para el diseño y desarrollo de la aplicación que pretenden ser los más útiles para su correcto funcionamiento y conseguir medir el rendimiento de la misma.

Además, también se detallará en este capítulo la plataforma donde se ha desarrollado y probado esta aplicación, las versiones de las tecnologías utilizadas, y demás consideraciones necesarias, que junto con los Apéndices que se incluyen al final del documento, en el que se detallan las configuraciones e instalaciones específicas de ciertas partes de la aplicación, pretende facilitar la reproducción del entorno en el cual se ha desplegado para poder montarse en cualquier momento.

El aspecto de la aplicación en funcionamiento se muestra en la siguiente ilustración, donde se aprecian los dos PCs (*Personal Computer*) que se comunican entre sí gracias al paradigma de publicación-suscripción. En ambos existe una comunicación bidireccional de mensajes de texto y una comunicación unidireccional de los datos de vídeo desde el PC 1 al PC 2.

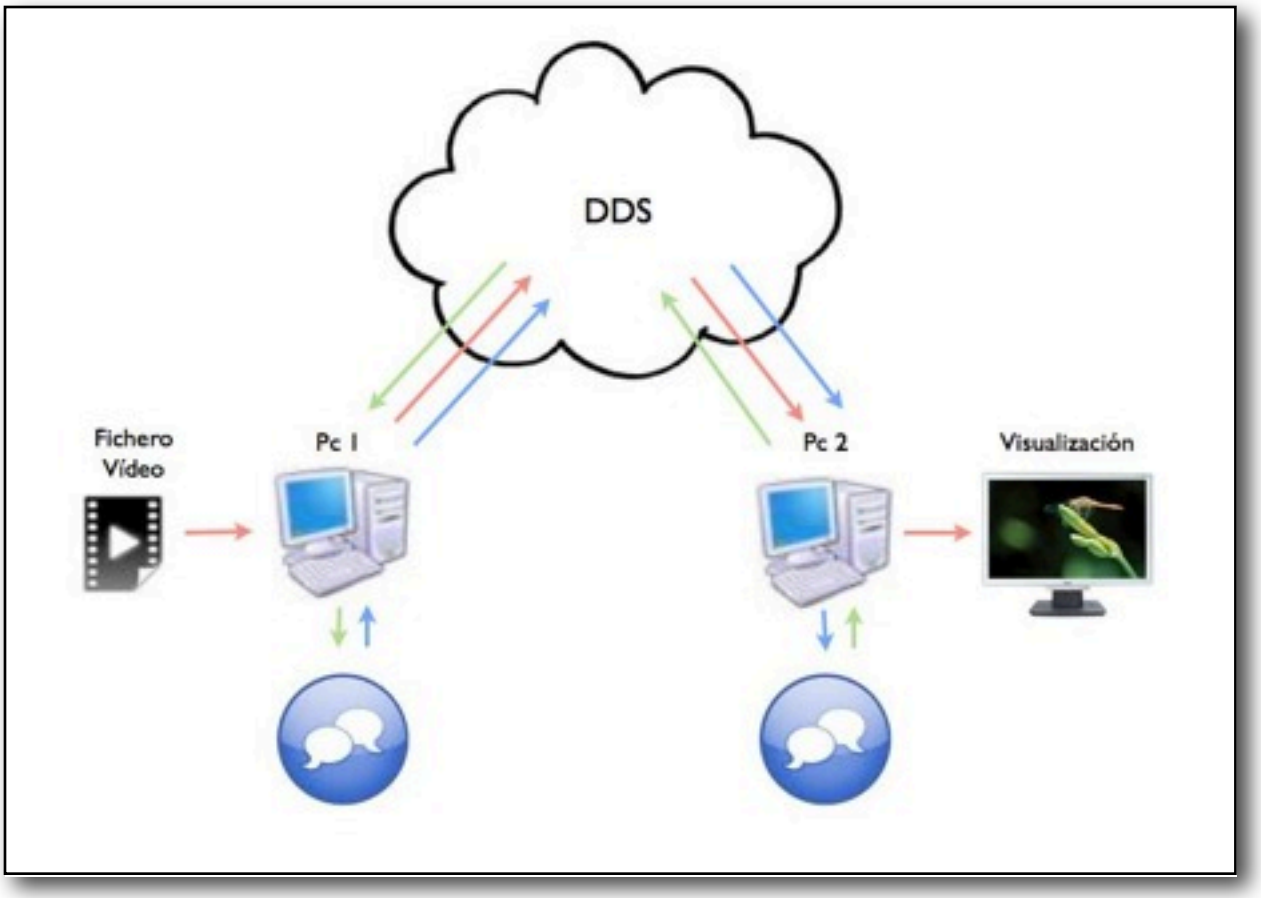


Ilustración 21. Visión global de la aplicación Vídeo-Chat

En la *Ilustración 21*, se puede observar la interacción que tienen los ordenadores:

- El PC 1 es el encargado de enviar los datos de un vídeo sobre el sistema de distribución. Además es capaz de tener una comunicación con el PC 2 por medio de mensajes de texto que se envían mutuamente.

- El PC 2 es el encargado de recibir los datos de vídeo y tratarlos para su reproducción. También posee la capacidad de comunicarse con el otro ordenador gracias a los mensajes de texto que se pueden enviar.

4.2 Entorno de Desarrollo

El entorno de desarrollo utilizado para la aplicación se ha desarrollado dentro de una maquina virtual Linux sobre la plataforma VirtualBox[3] en un Mac Book Pro con un procesador de 2.3 Ghz Intel Core i5, una memoria RAM de 4Gb 1333Mhz DDR3 y el sistema operativo Mac OS X 10.6.7.

Como se ha dicho antes, la máquina virtual se ejecuta sobre la aplicación Virtual Box, con un sistema operativo Ubuntu[4] y una memoria base de 1024 MB. Se evita la utilización de un sistema Windows por la versatilidad que ofrece un sistema Linux en cuanto a configuraciones y por la mayor gama de tecnologías y lenguajes que pueden utilizarse en éste.

4.3 Arquitectura y Diseño de la aplicación

Desde un principio la idea básica de la aplicación ha sido la transmisión de vídeo mediante DDS añadiéndosele la funcionalidad de envío de mensajes de texto al estilo de un Chat para la comunicación entre los publicadores y los subscriptores.

A continuación se detallan los requisitos necesarios para la realización de la aplicación, clasificados según sea su naturaleza, funcional o no funcional, comprobando la consecución de los mismos en el *apartado 5.6*, tras la realización de unas pruebas.

[3] La aplicación VirtualBox se puede descargar en https://www.virtualbox.org/

[4] La versión de Linux Ubuntu se puede descargar en http://www.ubuntu.com/download/ubuntu/download

Requisitos Funcionales

RF1: el sistema a desarrollar debe ser capaz de transmitir información en tiempo real.

RF2: el sistema debe visualizar contenidos multimedia.

RF3: el sistema debe minimizar el ancho de banda ocupado con el fin de no saturar la red, es decir, únicamente debe utilizar el necesario.

RF4: el sistema no tiene que producir una descarga completa del medio, tan solo descargar las partes utilizadas en dicho instante para la reproducción.

RF5: el sistema debe comunicar remotamente a los usuarios.

RF6: debe existir un intercambio de información entre los distintos componentes a través de los medios de comunicación que el sistema distribuido proporciona a cada componente.

RF7: la aplicación debe ofrecer la posibilidad de transmitir imágenes desde un fichero de vídeo y mostrarlas en pantalla.

RF8: la aplicación debe ofrecer la posibilidad de guardar el vídeo en disco.

RF9: la aplicación debe ofrecer la posibilidad de transmitir y recibir mensajes de texto de forma simultánea al envío y recepción del vídeo.

RF10: la aplicación debe dar la posibilidad de ser configurable por parte del usuario

Requisitos No Funcionales

RNF1: el sistema debe esconder la heterogeneidad de las plataformas que comunican.

RNF2: el sistema debe tener la mínima tolerancia a fallos.

RNF3: el sistema debe abstraer las aplicaciones distribuidas de redes, lenguajes de programación y sistemas operativos.

RNF4: el sistema debe aportar transparencia de localización, de espacio de nombres, de identificación, de redundancia, de acceso local o remoto, temporal o de errores.

RNF5: debe existir coordinación entre los distintos componentes del sistema distribuido.

RNF6: los componentes del sistema deben realizar diversas tareas de manera independiente y servirse de forma concurrente de los recursos que tienen a su disposición.

RNF7: los componentes del sistema deben conocer los protocolos de comunicación.

RNF8: el sistema distribuido debe ser heterogéneo, extensible, seguro, escalable, tolerante a fallos, concurrente y transparente.

RNF9: se deben conocer los parámetro de calidad de servicios que utiliza un sistema distribuido, entre ellos: plazo, orden en el destino, durabilidad, historia, etc.

RNF10: el sistema debe utilizar un modelo de publicación-suscripción.

RNF11: la arquitectura de publicación-suscripción debe estar formada por los siguientes componentes: productor de información, consumidor de información, mediador, canal, tópico, escritor y lector.

RNF12: el sistema debe tener un mínimo de dos clientes para el correcto funcionamiento.

RNF13: el sistema debe disponer de los códecs necesarios para la correcta visualización de los vídeos.

RNF14: el sistema debe soportar varios tipos de contenedores, como pueden ser: AVI o MPEG.

RNF15: el sistema puede ser desarrollado para ser utilizado en dispositivo portátil, fijo o móvil.

RNF16: la visualización del vídeo debe realizarse de forma continua y sin cortes.

RNF17: el sistema debe ser capaz de enviar y recuperar todos los datos de vídeo.

RNF18: el sistema debe ser capaz de enviar y recuperar todos los mensajes de texto.

RNF19: los mensajes de texto y los paquetes de vídeo deberán tener una longitud determinada para evitar problemas de sobrecarga.

RNF20: los mensajes de texto deben llegar en el mismo orden en el que se enviaron a nivel de aplicación.

RNF21: los paquetes de datos del vídeo deben llegar en el mismo orden en el que se enviaron a nivel de aplicación.

RNF22: la visualización del vídeo debe comenzar cuando se tengan datos suficientes, evitando así los cortes en la reproducción.

RNF23: en caso de falta de datos, la aplicación debe mantenerse a la espera de la llegada de más.

RNF24: el sistema de vídeo y de chat deben ser independientes.

Requisitos Mínimos

RM1: para que la aplicación funcione correctamente se debe disponer de un equipo con un procesador de 2.3ghz, procesador intel core i5, memoria Ram de 4gb.

RM2: se recomienda un sistema Linux para el uso de la aplicación, debido a la gran versatilidad que ofrece.

RM3: se debe disponer de una versión mínima 4.3 de OpenSplice instalada.

Para la resolución de estos requisitos previos, se estudió la posibilidad de incorporar en un mismo tópico el envío de los mensajes de texto y los datos de vídeo, rechazándola posteriormente, al determinar la necesidad de independencia de ambos tipos de datos, para la recepción y emisión de los mismos.

Por todo ello, y por la utilización de un *middleware* distribuido, la arquitectura de la aplicación corresponde al clásico publicador-subscriptor, en el que un mínimo de dos nodos van a intercambiar mensajes correspondientes a uno o varios tópicos comunes a los que están subscritos y en los que se están publicando datos.

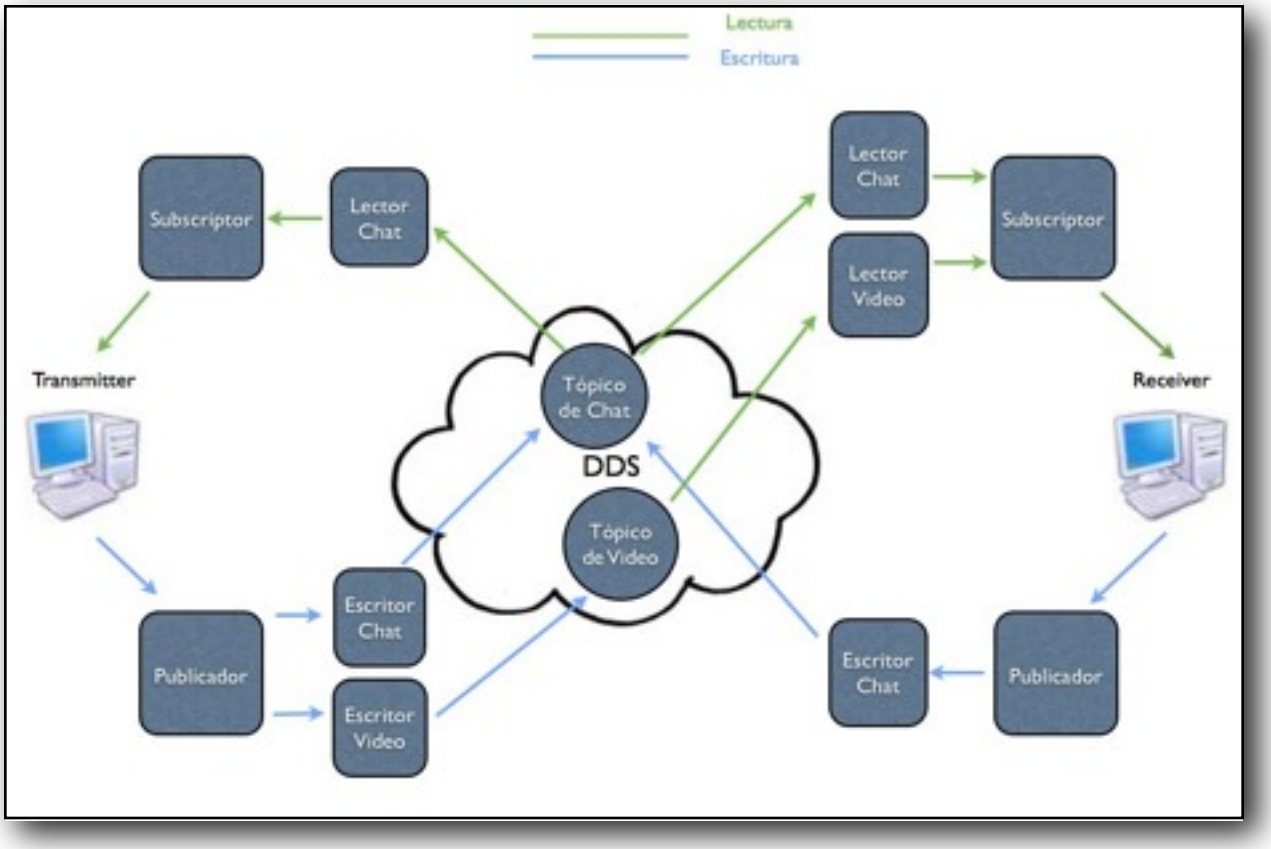


Ilustración 22. Diagrama de arquitectura de la aplicación

Como se muestra en la *Ilustración 22*, ambos agentes (Transmitter y Receiver) son publicadores y subscriptores dependiendo del tópico al que estén asociados.

- **Transmitter**: es el encargado de publicar los datos del vídeo y de chat dentro de su tópico correspondiente, además estará subscrito a los mensajes que se escriban dentro del tópico del chat.

- **Receiver**: es el encargado de recibir los datos de vídeo, para su posterior reproducción, y los mensajes de texto. Además, publicará mensajes de chat.

La estructura interna de la aplicación está dividida en dos agentes independientes (Transmitter y Receiver), donde en cada uno de ellos se implementan 3 módulos básicos de ejecución en los que se realizan todas las funcionalidades de la aplicación. En un primero momento, la estructura estaba formada únicamente por un módulo que realizaba todas las funcionalidades, esto producía que la aplicación no fuese limpia y clara. Por ello, manteniendo los dos agentes, las funciones que realizaban cada uno de ellos se fue incorporando en módulos diferentes para así mantener un diseño y una estructura mejorada.

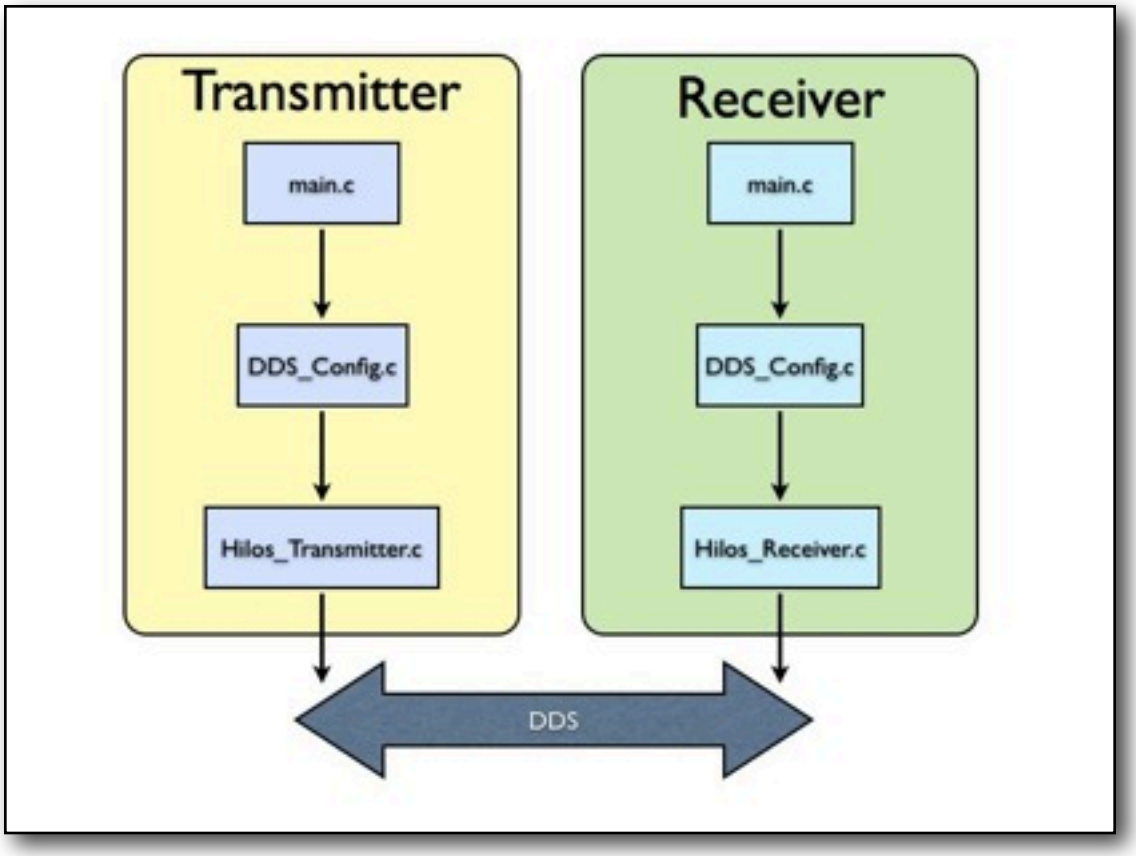


Ilustración 23. Estructura interna de la aplicación

Como se muestra en la *Ilustración 23*, existen dos agentes que se comunican a través de DDS. Esta figura es una abstracción de la aplicación ya que cada agente posee varias funciones que se verán más adelante. A continuación se va a comentar cada una de las funciones que se realizan en cada uno de los nodos.

- **Transmitter**: el nodo transmisor está compuesto por tres módulos que realizan la funciones necesarias para la inicialización de la aplicación y publicación de mensajes (main.c), configuración del entorno DDS (DDS_Config.c) y publicación de vídeo y recepción de mensajes (Hilos_Transmitter.c).

- **Receiver**: el nodo receptor está compuesto, como en el caso anterior, por tres módulos que realizan las funciones necesarias para la inicialización de la aplicación y recepción de mensajes (main.c), configuración del entorno DDS (DDS_Config.c) y recepción de vídeo y publicación de mensajes (Hilos_Receiver.c). A diferencia que en el caso anterior, este posee un mayor conjunto de hilos sincronizados, para la correcta reproducción de los datos de vídeo. Este diseño está pensado para que un hilo pueda seguir recibiendo datos mientras el otro hilo los va reproduciendo, evitando así la carga computacional que acarrearía realizar ambas operaciones en el mismo hilo.

El momento de la obtención de los datos de vídeo y su reproducción es el más crítico de toda la aplicación, ya que se deben sincronizar los datos recibidos con los datos a reproducir. El reproductor de vídeo no puede pedir datos mientras estos no hayan sido recibidos, por lo que ambos hilos deben ser sincronizados utilizando el paradigma de productor-consumidor.

En un primer momento, se pensó en la espera, por parte del reproductor, de esos datos mientras eran introducidos en un buffer auxiliar en el que se iban acumulando hasta que estaba completamente lleno. Una vez lleno, el reproductor consumía esos datos mientras el subscriptor del vídeo seguía recibiendo paquetes de datos, como se muestra en la *Ilustración 24*.

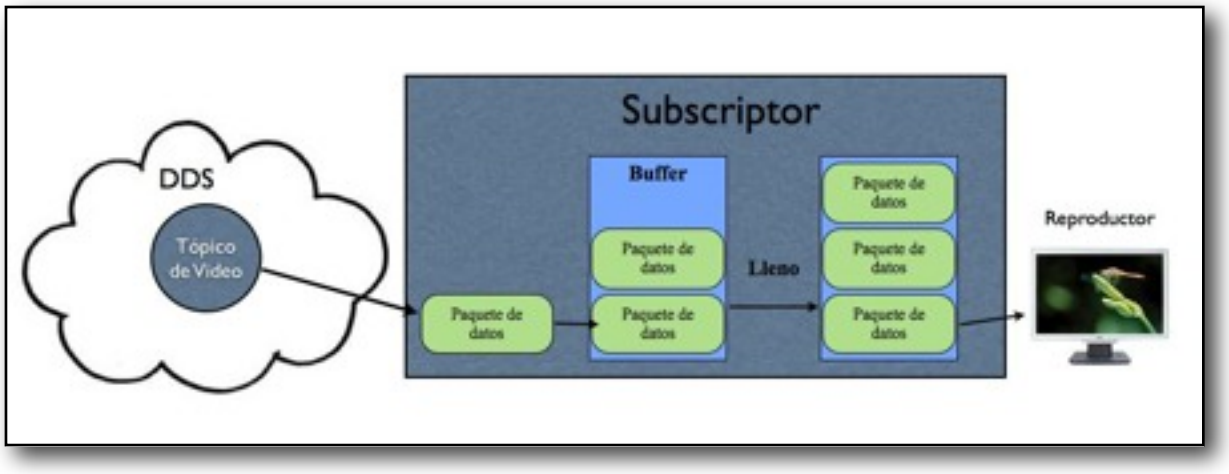


Ilustración 24. Proceso de llenado del Buffer

Este diseño produce un retardo en la reproducción del vídeo debido a que el reproductor consume paquetes de datos mucho más rápido que la velocidad a la que se reciben y se almacenan esos paquetes.

Por ello, se tuvo que rediseñar la parte de almacenamiento y reproducción introduciendo una Fila en la que se introducen los buffers, de un tamaño definido, que contiene un número determinado de paquetes de datos recibidos. A su vez, se diseña el reproductor para que se mantenga a la espera mientras no exista un mínimo de paquetes dentro de la Fila, y así evitar que se produzcan cortes en la reproducción del vídeo, como se muestra en la *Ilustración 25*. Gracias a este cambio, la reproducción del vídeo se realiza de forma más fluida evitando la espera de la llegada de paquetes de datos.

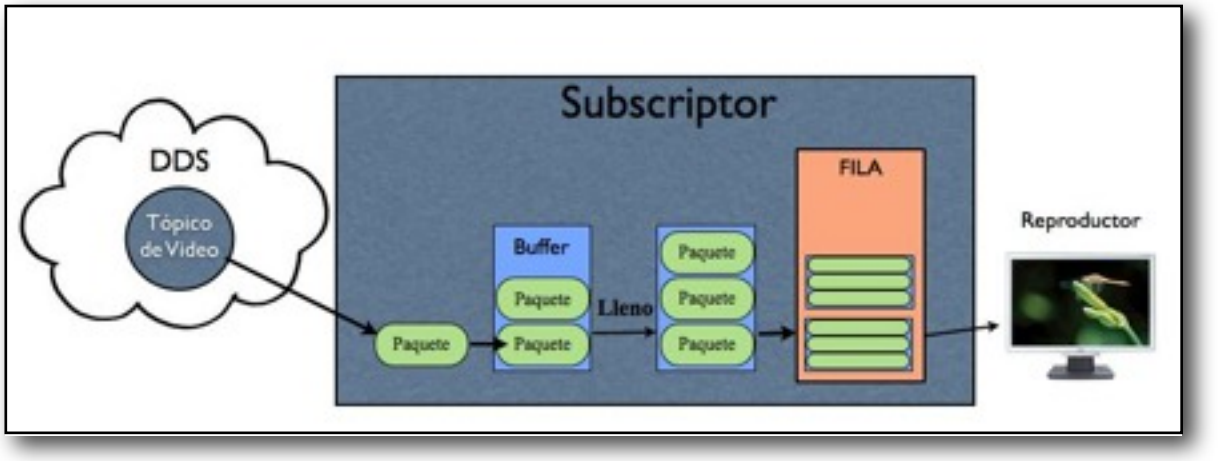


Ilustración 25. Reproducción mediante una Fila intermedia

4.4 Desarrollo de la aplicación

En este apartado se va a comentar como se ha desarrollado la aplicación, encontrando en la misma tres partes bien diferenciadas. En la primera parte, se habla del modelado de datos comunes a la aplicación realizado mediante IDL, en la segunda se detalla la función del emisor "Transmitter" del vídeo y por último, en la tercera parte, las funciones del receptor "Receiver". Cabe destacar que tanto el emisor como el receptor son publicadores y subscriptores del tópico del Chat, por el cual se comunican, pero únicamente el emisor es publicador del vídeo y el receptor, subscriptor del mismo, de ahí sus nombres. La *Ilustración 26* muestra un diagrama simple del desarrollo.

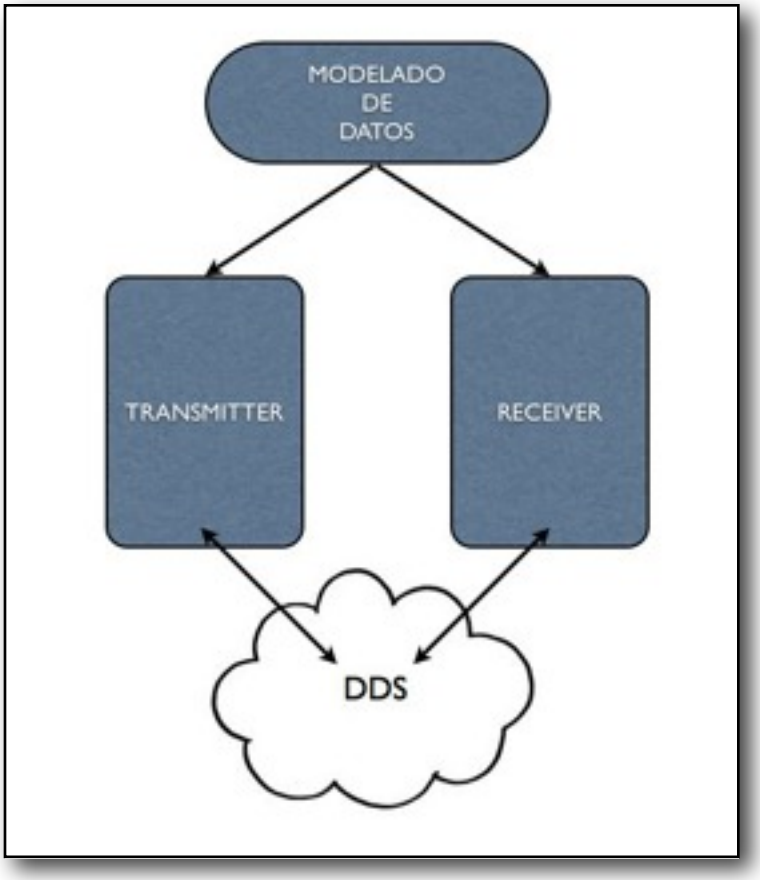


Ilustración 26. Diagrama de desarrollo

4.4.1 Modelado de datos

OpenSplice DDS distribuye sus datos en tipos estructurados, que son transportados por medio de tópicos "*topics*". El primer paso cuando se utiliza OpenSplice consiste en definir estos tipos de datos. Dado que OpenSplice se puede utilizar en varias plataformas diferentes, con varios lenguajes de programación diferentes, se utiliza IDL como lenguaje y plataforma de modelado independiente. Todos los datos que se quieran distribuir mediante DDS tienen que estar definidos como tópicos con un tipo estructurado de datos, unos "datos clave" definidos mediante IDL.

Por todo ello, para la generación de los tópicos y constantes comunes de la aplicación se utilizan estos datos que se encuentra definidos dentro de un fichero con extensión "*.idl*". Para ello se genera una estructura de datos con diferentes miembros y datos clave:

```
struct ChatMessage {
        string    chatterName;      // Propietario del Mensaje
        long      index;            // Número del Mensaje
        string    content;          // Contenido del Mensaje
};
```

La definición de cada uno de los datos que van a ser transmitidos, tienen que estar definidos en este archivo IDL.

Para la aplicación que se ha realizado, como se muestra en la *Ilustración 27*, se tienen que definir dos estructuras diferenciadas, correspondientes a los tópicos que se van a publicar:

1. Un tópico referente a los mensajes de texto que se van a intercambiar los participantes.

2. Un tópico que corresponde a los datos de vídeo que se van a transmitir desde un publicador a los diferentes subscriptores.

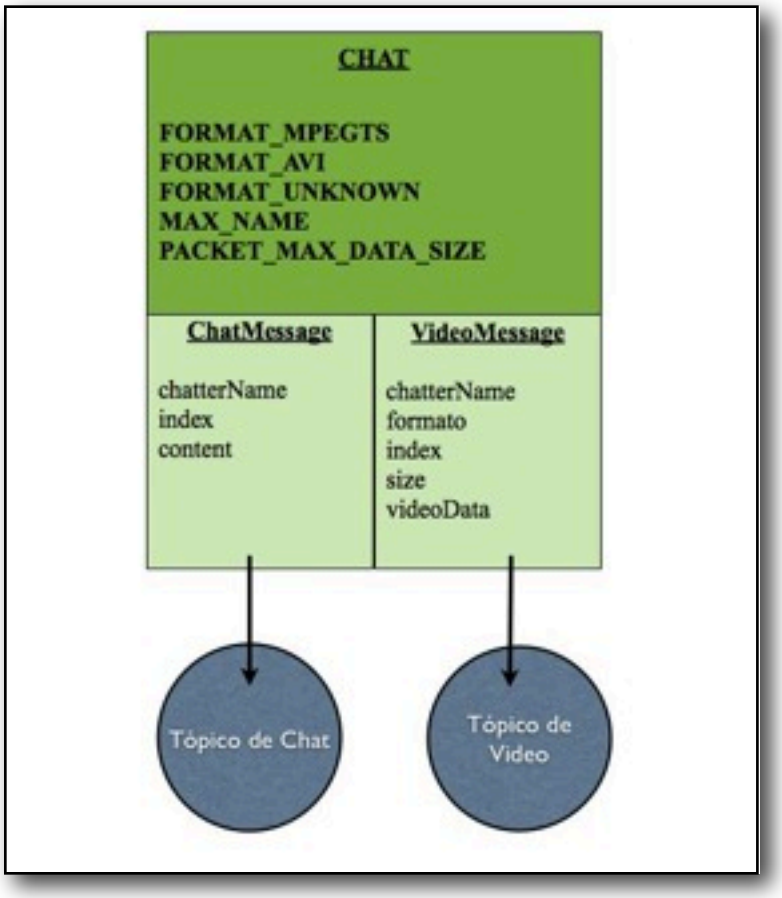


Ilustración 27. Modelo IDL de datos.

Ambos tópicos van a tener un campo clave, el nombre de usuario, por el cual será identificado el participante dentro del dominio de distribución. Ambas estructuras pertenecen a un módulo superior que las une para la posterior obtención de los datos. En este caso el módulo tiene el nombre de "Chat".

Tópico Chat

Para la estructura de los mensajes de texto que se van a intercambiar, se han definido tres campos:

- chatterName: el primero de ellos corresponde al nombre del usuario que emite el mensaje, este dato, como se ha dicho anteriormente, corresponde a un campo clave para poder identificar a los usuarios.
- index: el segundo corresponde al número de mensaje que está siendo transmitido.
- content: el tercero de los datos es el contenido del mensaje, es decir, el texto que ha escrito el usuario.

Tópico Vídeo

Para la estructura de los mensajes de vídeo se han definido cinco campos que son necesarios para la correcta transmisión del vídeo:

- chatterName: como en el caso de los mensajes de texto, corresponde al nombre de usuario que esté publicando el vídeo.
- formato: este campo corresponde con el formato del vídeo que se está publicando, este dato es necesario para que el subscriptor pueda decodificar los datos y visualizar el contenido de los mismos.
- index: número de paquete de datos que se esté publicando, gracias a este dato se comprueba en la recepción de los datos si se han recibido todos correctamente.
- size: tamaño de los datos que se estén publicando.
- videoData: este campo corresponde a los datos de vídeo propiamente dichos.

En este fichero también se encuentran definidas unas constantes comunes que se presuponen necesarias para los emisores y receptores. En este caso corresponden al tamaño máximo que puede tener el nombre de usuario (**MAX_NAME**), el tamaño máximo de los paquetes de datos que se van a transmitir (**PACKET_MAX_DATA_SIZE**) y un mapeo que se realiza para identificar el tipo de formato de vídeo que se esté enviando (**FORMAT_X**).

A pesar de que el tipo de datos se define utilizando IDL, la aplicación va a utilizar una estructura C equivalente. Esto se logra mediante la invocación del preprocesador que el sistema OpenSplice posee:

```
$ idlpp -S -l c Chat.idl
```

Esta aplicación traduce la definición IDL de los tipos de datos en una definición de C correspondiente. El preprocesador altera el nombre de las estructuras añadiéndoles el prefijo del módulo al que pertenecen, para poder conocer la pertenencia de la estructura en concreto.

Además genera las funciones necesarias para la obtención, escritura y modificación de los datos de las estructuras dentro de la aplicación generando los correspondientes ficheros .h y .c que serán importados dentro de los agentes "Transmitter.c" y "Receiver.c".

4.4.2 Transmisor

El transmisor, como se comentó anteriormente, es el agente encargado de la publicación del vídeo y de los mensajes de texto, así como, la suscripción de los mensajes de texto publicados por otros usuarios. Para este cometido, como se muestra en la *Ilustración 28*, el agente consta de tres hilos que realizan cada una de las operaciones. Además, se realiza una configuración inicial por parte del usuario y de los elementos necesarios para la distribución de datos, como se verá más adelante.

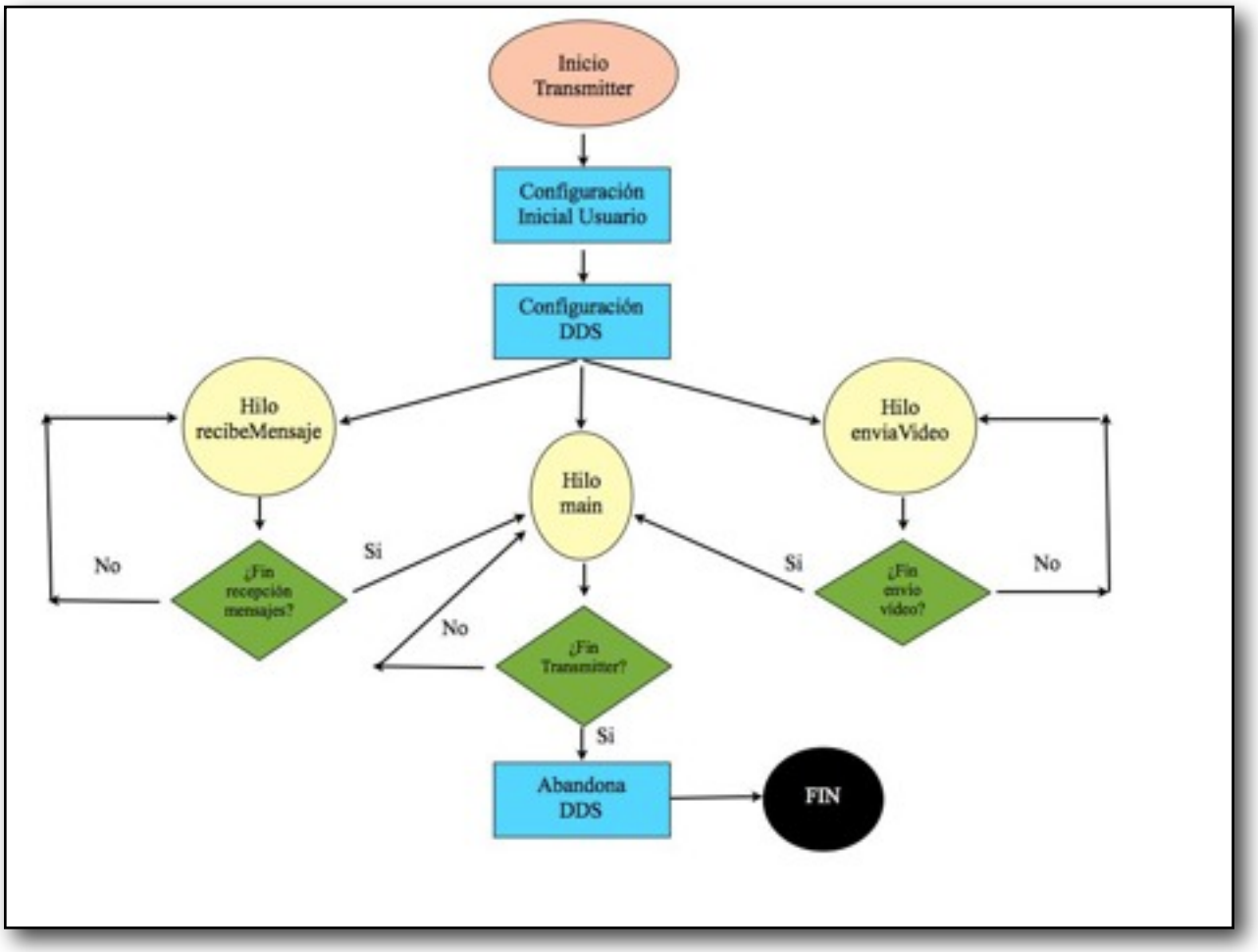


Ilustración 28. Arquitectura del Transmisor

A continuación se detalla cómo se ha realizado la configuración y las funciones de los hilos.

Configuración Inicial Usuario

Al iniciar la aplicación, el usuario puede configurar algunos parámetros para para modificar el funcionamiento de la misma. La configuración consta de 5 parámetros de los cuales dos son obligatorios y el resto tiene valores por defecto en el caso de que el usuario no desee modificarlos:

- *chatterName*: nombre del usuario que arranca la aplicación, será el nombre con el que envía los mensajes.

- *nombreVideo*: ruta al fichero de vídeo que se quiere publicar. Tiene que ser la ruta completa donde se encuentra el fichero (*/home/videos/Video.ts*).

- *chatTopicConfig*: configuración del chat:
  - 0 (*Live Chat*): se recibe el último mensaje enviado al Chat y los siguientes que se envíen (Opción por defecto si se pulsa intro).
  - 1 (*Tweet Chat*): se reciben mensajes anteriores a estar conectado.

- *chatTopicName*: nombre del tópico de chat al que se va a conectar, (Por defecto será 'Chat').

- *videoTopicName*: nombre del tópico de vídeo al que se va a conectar, (Por defecto será 'Vídeo').

Esta configuración inicial permite saber qué usuario está enviando los mensajes de texto, seleccionar el vídeo a publicar, configurar cómo se quieren recibir los mensajes de texto y seleccionar el tópico de Chat en el que se va a conectar, así como el tópico de vídeo en el que se enviará la película.

Configuración DDS

Tras la configuración inicial del usuario se debe inicializar la configuración DDS con los parámetros elegidos anteriormente. La creación de los elementos es idéntica a cómo se realizó en el *Paso 1 de la Aplicación Inicial del Capítulo 3*, únicamente difiere en los siguientes elementos:

- **Tópicos**: se necesitan dos tópicos, uno para el vídeo y otro para los mensajes cuyos nombres vienen especificados en la *Configuración Inicial de Usuario*, con una configuración de QoS por defecto en la que los mensajes sean fiables, se entreguen en orden y se mantengan en el entorno mientras este esté activo.

- **Publicadores**: se utiliza el mismo publicador para los escritores con las misma QoS que los tópicos.
- **Escritores**: se necesitan dos escritores de datos, uno para el tópico de vídeo y otro para el chat.
- **Subscriptores**: se necesita un único subscriptor para los datos de los mensajes de texto.
- **Lectores**: se necesita un lector para los mensajes de chat. Este lector se configurará con el parámetro *chatTopicConfig* que se especificó en la *Configuración Inicial de Usuario*.

Finalizado este proceso, el usuario que arranca el transmisor es capaz de publicar mensajes de texto, así como, datos del vídeo que seleccione; y subscribirse a los mensajes de texto enviados por otros usuario.

HILOS

Para el correcto funcionamiento de la aplicación transmisora, las diferentes acciones que se pueden realizar se han dividido en tres hilos ("*recibeMensaje*", "*enviaVideo*", "*main*") para una mejor independencia y modularidad del sistema. Cada uno de estos hilos son ejecutados tras las configuraciones y ejecutan una funcionalidad diferente que se explica a continuación:

- **recibeMensaje**: este hilo es el encargado de obtener los mensajes publicados por otros usuarios. Para ello posee un subscriptor, creado previamente en la inicialización DDS, el cual lee los mensajes que están escritos sobre el tópico de Chat. Este hilo, como se muestra en la *Ilustración 29*, se mantiene en ejecución mientras se estén enviando mensajes y finaliza únicamente cuando se obtiene un mensaje de finalización de chat (***TERMINATION_CHAT***) o el propio usuario cierra la ventana de mensajes.

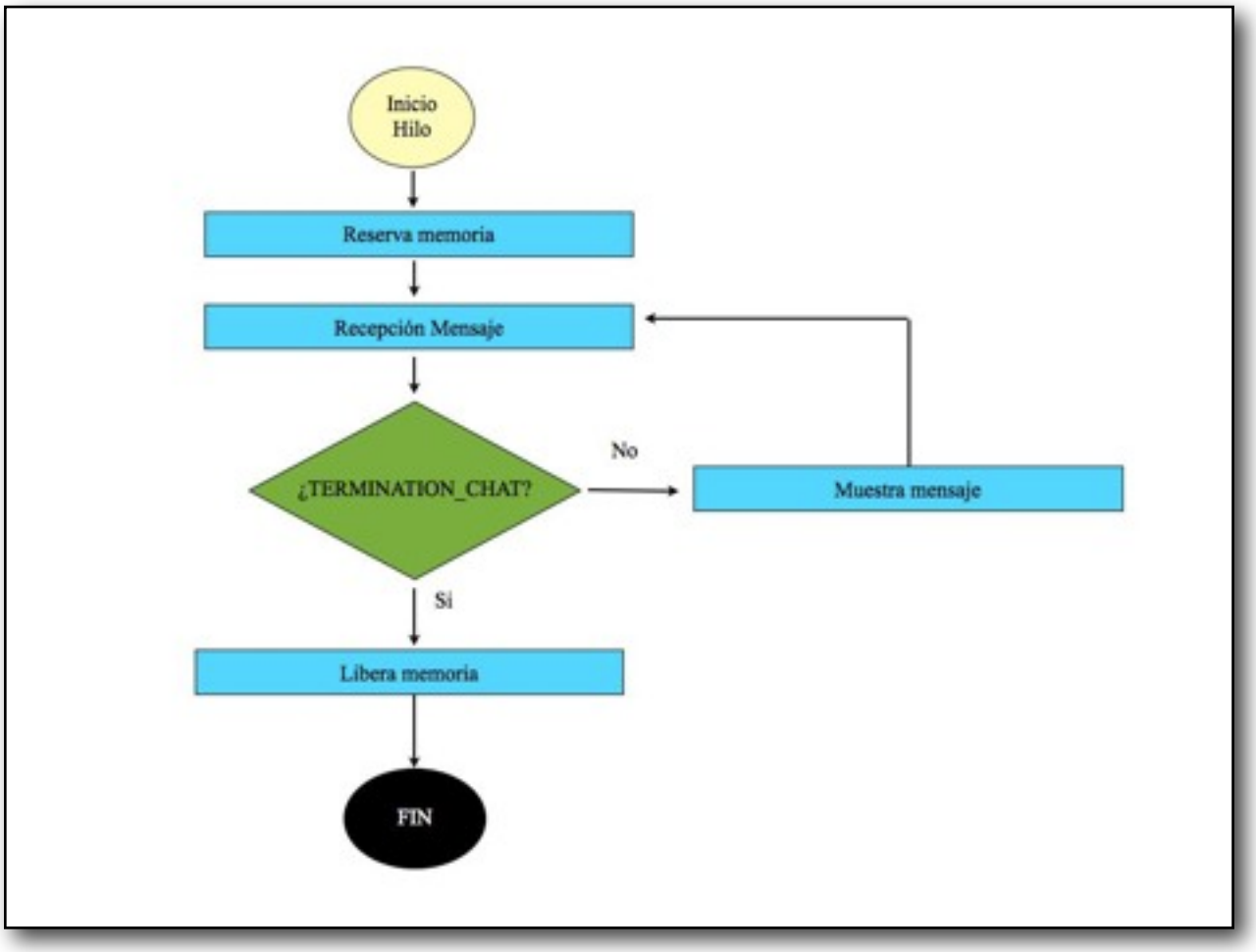


Ilustración 29. Diagrama de estados del hilo recibeMensaje

- **enviaVideo**: este hilo es el encargado de escribir los datos del vídeo que se quiere publicar. La primera acción que realiza es la apertura del fichero de vídeo que se quiere enviar, dicho vídeo ha sido seleccionado por el usuario en la *Configuración Inicial de Usuario*. Los datos del fichero son leídos, 2048 bytes en cada lectura[5], e introducidos dentro de los campos de la estructura de vídeo para su posterior publicación. El campo del formato del vídeo es obtenido a partir de la extensión del fichero. Como se ha dicho anteriormente, este campo será necesario para la correcta decodificación y reproducción del vídeo.

[5] Se comprobó que era un tamaño suficiente para que el envío del paquete no fuese muy grande y contuviese la suficiente información para que la reproducción se realizase lo antes posible.

Otro dato importante que se publica es el número de paquete que se esté enviando, gracias a él se puede realizar el seguimiento de los paquetes que son recibidos. Una vez conseguidos los datos que se quieren publicar se deben registrar en el escritor para su posterior escritura en el dominio de participación. La finalización de este hilo se produce cuando se realiza el envío del último paquete de datos del fichero de vídeo. En ese momento, además del paquete, se envía un código (**TERMINATION_VIDEO**) que permite conocer a los subscriptores que el envío de datos ha terminado, se liberan los recursos reservados y se cierra el fichero de vídeo. El diagrama de estados del hilo se muestra en la *Ilustración 30*.

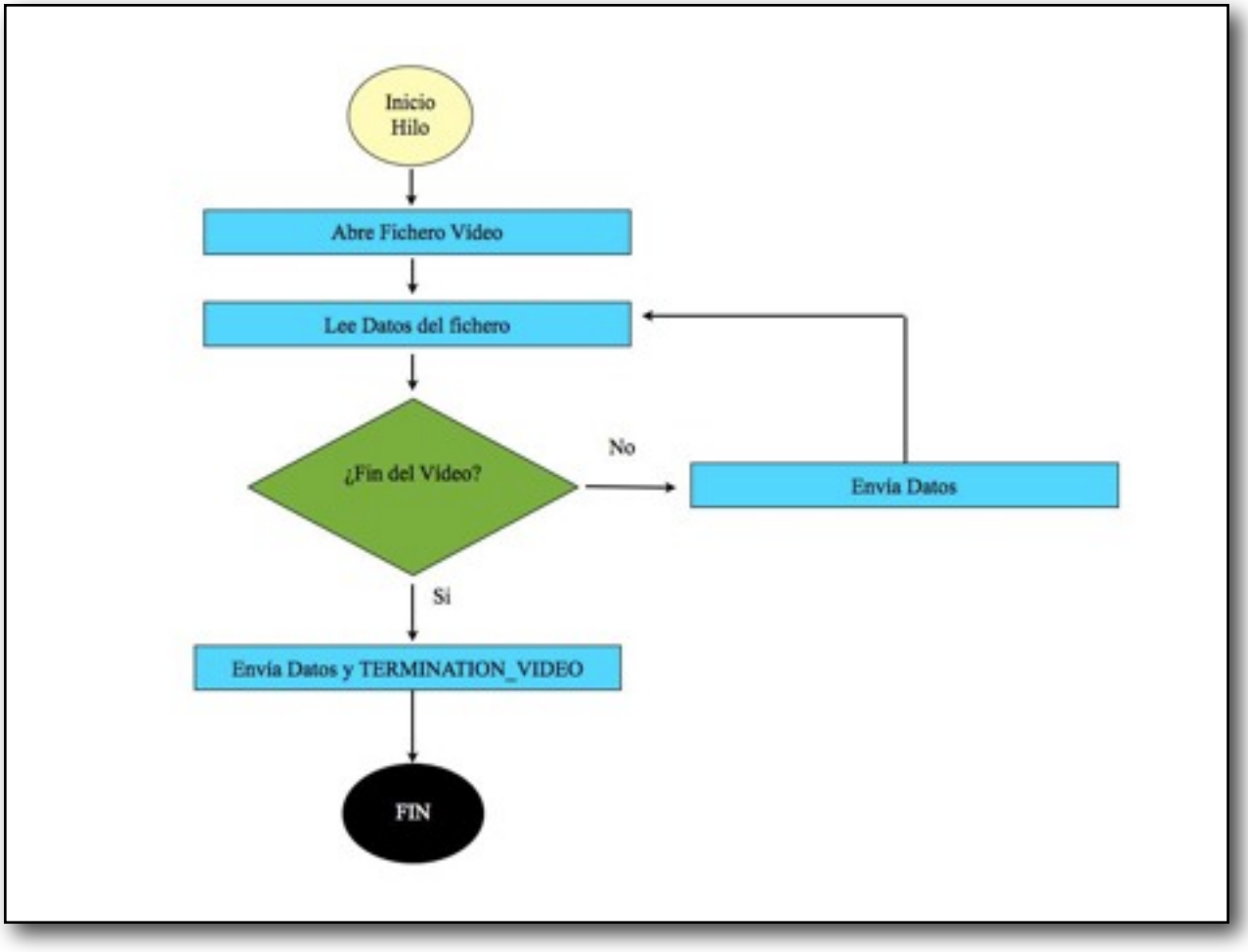


Ilustración 30. Diagrama de estados del hilo enviaVideo

- **main**: este hilo es el principal de la aplicación de transmisión. Es el encargado de la inicialización y el liberado de datos de los elementos necesarios para la distribución de datos, así como, de la creación de los hilos anteriores. Otra de las funciones que realiza es la que se muestra en la *Ilustración 31*, publicar los mensajes de texto que el usuario de la aplicación quiere transmitir. Para ello, de forma muy similar a los hilos anteriores, obtiene el mensaje de texto y el escritor los envía al dominio de participación en su tópico correspondiente. Este proceso es repetido hasta que el usuario decide abandonar la aplicación.

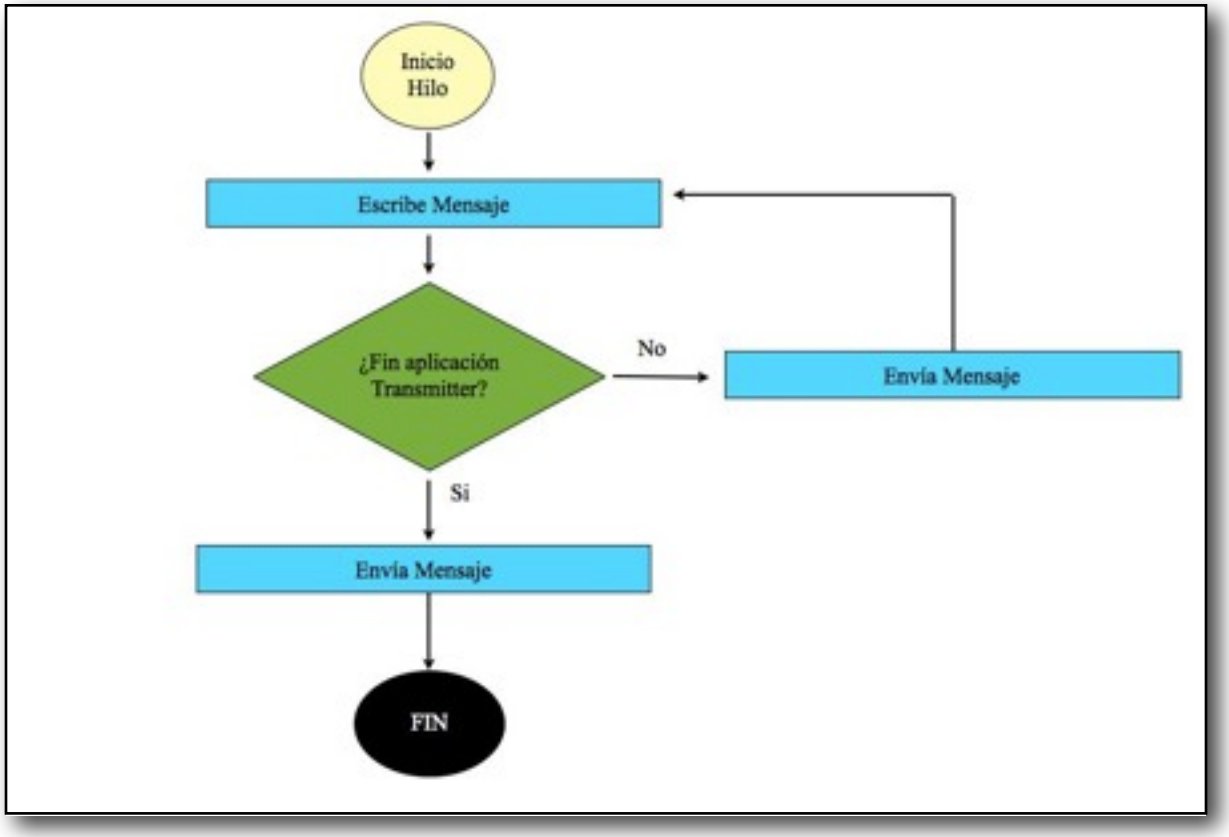


Ilustración 31. Diagrama de estados del hilo main

Abandono DDS

Una vez, el usuario decide terminar, la aplicación libera los recursos de los hilos y estructuras que han sido creadas y abandona la dominio de participación DDS creado al inicio. Para abandonar este dominio debe de hacerse de forma similar a cómo se explicó en el *Paso 3 de la Aplicación Inicial del Capítulo 3*, pero con los elementos creados en ésta.

4.4.3 Receptor

La segunda parte de la aplicación la lleva a cabo el Receptor o *Receiver*. Este agente es el encargado de la recepción, reproducción y visualización del vídeo que es publicado por la parte transmisora. A su vez, también tiene la capacidad de publicar mensajes de texto para comunicarse con el resto de agentes que estén conectados al dominio de distribución, así como, estar subscrito a otros mensajes publicados en dicho dominio.

Este agente es el más complejo en cuanto a lógica de programación, ya que para un correcto funcionamiento de las acciones que realiza, se han tenido que arrancar diversos hilos para poder trabajar en paralelo y sincronizar las actividades que realizan unos con otros. En la *Ilustración 32*, se muestra la arquitectura que tiene la aplicación Receptora.

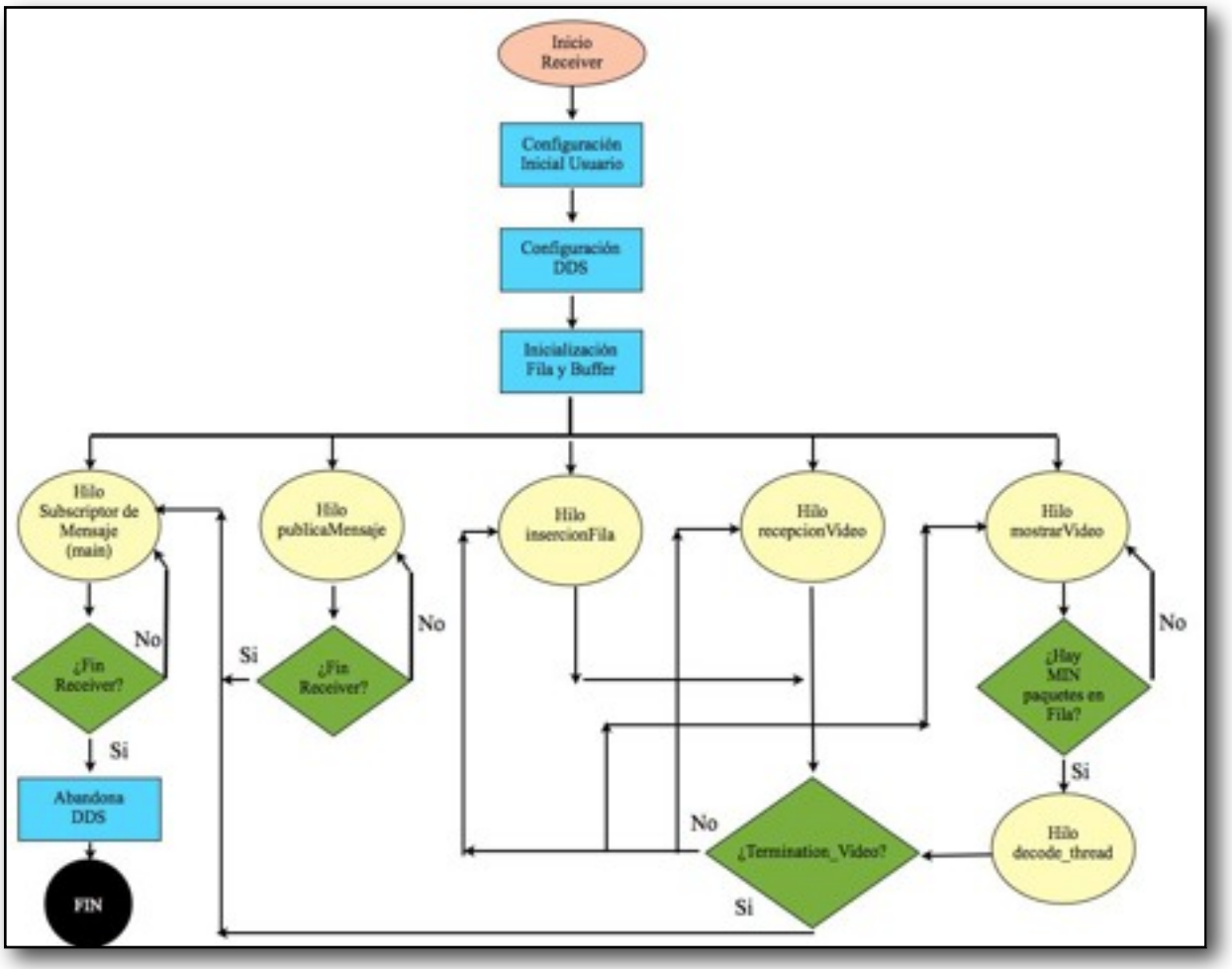


Ilustración 32. Arquitectura del Receptor

La estructura del receptor se puede dividir en 10 partes en la que alguna de ellas inician varios hilos que realizan un trabajo en común de forma síncrona:

1. Configuración Inicial del Usuario.
2. Inicialización de la configuración DDS
3. Inicialización de la Fila y del Buffer de datos recibidos
4. Inicialización del Subscriptor de Mensajes de texto
5. Inicialización del Publicador de Mensajes de texto
6. Inicialización del Receptor del Vídeo
7. Inicialización del Insertor de datos del Vídeo en la Fila
8. Inicialización del Reproductor del Vídeo
9. Inicialización del Visualizador del Vídeo
10. Abandono DDS y liberación de los recursos obtenidos

A continuación se explicará con más detalle las partes más relevantes de las que consta este agente.

Configuración Inicial Usuario

Al iniciar la aplicación, el usuario puede configurar algunos parámetros para modificar el funcionamiento de la misma. La configuración consta de 6 parámetros de los cuales únicamente el nombre del usuario es obligatorio y el resto tiene valores por defecto en el caso de que el usuario no desee modificarlos:

- *chatterName*: nombre del usuario que arranca la aplicación, será el nombre con el que envía los mensajes.

- *chatTopicConfig*: configuración del chat:
    - 0 (*Live Chat*): se recibe el último mensaje enviado al Chat y los siguientes que se envíen (Opción por defecto si se pulsa intro).
    - 1 (*Tweet Chat*): se reciben mensajes anteriores a estar conectado.

- *chatTopicName*: nombre del tópico de chat al que el usuario se va a conectar, (Por defecto será 'Chat').

- *videoTopicConfig*: configuración del vídeo:
    - 0 (*Live Stream*): se recibe el vídeo en directo (Opción por defecto si se pulsa intro).
    - 1 (*VoD*): se recibe el vídeo completo desde el incio.

- *videoTopicName*: nombre del tópico de vídeo al que el usuario se va a conectar. (Por defecto será 'Vídeo').

- *ficheroRecibido*: ruta y nombre donde se va a guardar el fichero recibido. Tiene que ser la ruta completa donde se encuentra el fichero (*/home/videos/Video.ts*). Por defecto, el vídeo no se guardará si no se establece una ruta.

Esta configuración inicial permite saber qué usuario está enviando los mensajes de texto, configurar cómo se quieren recibir los mensajes de texto y seleccionar el tópico de Chat en el que se va a conectar, así como el modo de reproducción del vídeo, el tópico de vídeo del que se quiere recibir la película y si se guarda un fichero con los datos.

Configuración DDS

Tras la configuración inicial del usuario se debe inicializar la configuración DDS con los parámetros elegidos anteriormente. La creación de los elementos es idéntica a cómo se realizó en el *Paso 1 de la Aplicación Inicial del Capítulo 3* y en el *Transmisor*, únicamente difiere en los siguientes elementos:

- **Tópicos**: se necesitan dos tópicos, uno para el vídeo y otro para los mensajes cuyos nombres vienen especificados en la *Configuración Inicial de Usuario*, con una configuración de QoS por defecto en la que los mensajes sean fiables, se entreguen en orden y se mantengan en el entorno mientras este esté activo.

- **Escritores**: se necesita un escritor de datos para los mensajes de chat.

- **Subscriptores**: se necesitan dos subscriptores, uno para los datos de los mensajes de texto y otro para los datos de vídeo.

- **Lectores**: se necesita un lector para los mensajes de chat que se configurará con el parámetro *chatTopicConfig* que se especificó en la *Configuración Inicial de Usuario*, y otro para los datos de vídeo que estará configurado con *videoTopicConfig*.

Inicialización de la Fila y del Buffer

En el receptor, se debe inicializar una estructura de datos común que corresponde a una "*Fila*" en la que se van a ir introduciendo los datos de vídeo recibidos en paquetes de un tamaño definido para su posterior obtención a la hora de reproducirlos y visualizarlos. Estos paquetes se corresponden a Buffers de un tamaño algo superior a 1Mb (1236992bytes), para que el reproductor cuente con una cantidad de datos suficiente para poder iniciar la visualización del vídeo, como ya se mostró en la *Ilustración 25*.

Subscriptor de Mensajes

Al igual que en el Transmisor, es el hilo encargado de obtener los mensajes de texto que son publicados por otros usuarios y los muestra por pantalla. Este proceso se inicia dentro del hilo *main* que arranca el agente receptor. Para ello, como se mostró en la *Ilustración 29* del Transmisor, el lector reserva memoria para dichos mensajes, una vez obtenido uno, es mostrado por pantalla, acompañado del nombre del usuario que lo ha publicado y la memoria que ha sido reservada para el mensaje es liberada para la obtención de otro. Este hilo se mantiene en ejecución mientras se estén enviando mensajes y finaliza únicamente cuando se obtiene un mensaje de finalización de chat (***TERMINATION_CHAT***) o el propio usuario cierra la ventana de mensajes.

Publicador de Mensajes

Es el hilo encargado de escribir los mensajes que el usuario quiere publicar. Este proceso se arranca como un hilo independiente (*publicaMensajes*) al iniciar la aplicación. El hilo es inicializado junto con una estructura de datos que es necesaria para la publicación de los mensajes. La primera acción que realiza es el registro de la instancia que realiza la escritura (*talker*), así como del mensaje que se va a publicar en sí. Una vez realizado esto, el hilo se mantiene a la espera de que el usuario escriba un mensaje de un tamaño definido, en este caso 256 caracteres, en la pantalla para su posterior escritura y publicación en el dominio de distribución. El funcionamiento de este hilo es idéntico al de la *Ilustración 31* del Transmisor.

Recepción del Vídeo

La primera parte dentro de la lógica que interviene en el vídeo es la recepción de los datos que son publicados. Para ello la aplicación arranca un hilo (*recepcionVideo*) que es el encargado de este cometido. En su inicialización, el hilo obtiene, entre otros parámetros, el buffer común que comparten el hilo de Recepción y de Inserción y la estructura con la fila en la que se van a escribir los datos que van a compartir los procesos que estén involucrados: Recepción, Reproducción, Inserción y Visualización. Para un correcto funcionamiento, estos hilos están sincronizados a la hora de acceder a los paquetes de datos de la fila.

En primer lugar, como se muestra en la *Ilustración 33*, se crea un *buffer* auxiliar de un tamaño suficientemente grande para poder copiar los datos que el lector está obteniendo. Una vez el *buffer* auxiliar está lleno, se realiza una copia del mismo sobre otro del mismo tamaño, el buffer común, que es compartido por el hilo "*Insertor*" (posteriormente este hilo introducirá este paquete en la Fila común), y el buffer auxiliar se sigue rellenando con los datos del vídeo que continúan recibiéndose.

En un primer momento, este hilo y el "Insertor" eran el mismo, y la recepción de los datos y la introducción de los paquetes rellenos dentro de la Fila, eran realizados de una forma continua. Este sistema producía que mientras el hilo introducía los paquetes, los datos que estaban siendo recibidos, no eran guardados ya que el hilo estaba ocupado insertando paquetes en la Fila.

Por esta razón, se dividió el trabajo en dos hilos que trabajasen de forma paralela, como se muestra en la *Ilustración 33*, evitando así la pérdida masiva de paquetes de datos.

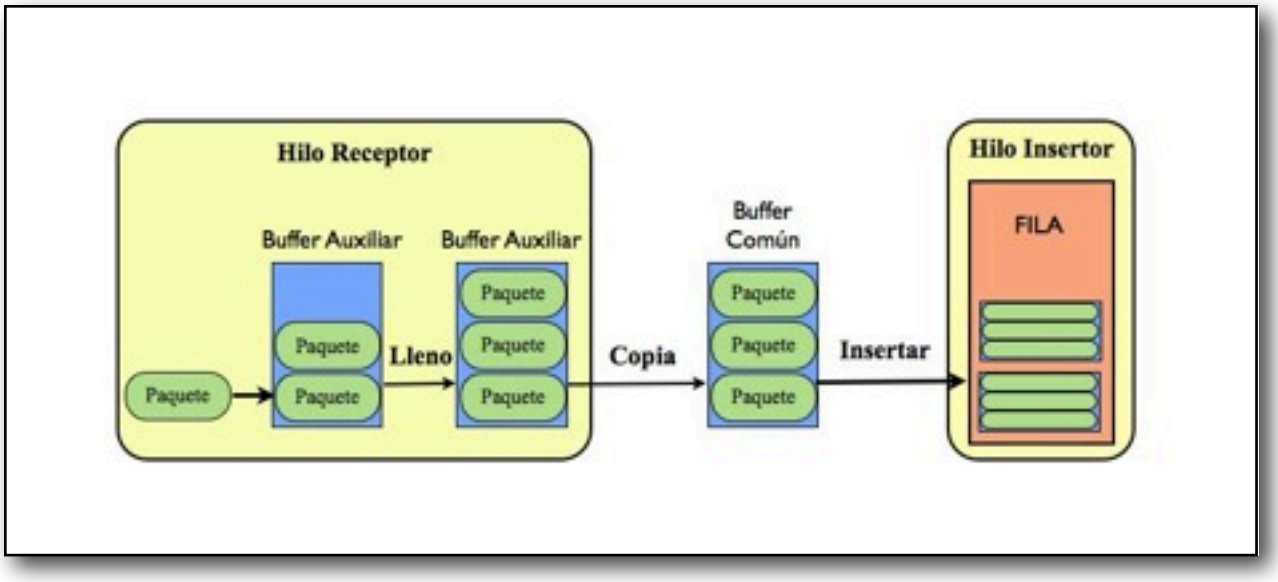


Ilustración 33. Proceso de Recepción

En la *Ilustración 34*, se observa como el proceso de recepción de datos y copia, es repetido hasta que se recibe el último paquete de datos por parte de Transmisor, que como se dijo anteriormente, viene indicado por el código **TERMINATION_VIDEO.** Una vez recibido, se realiza la copia de este último paquete en el buffer auxiliar y éste, a su vez, al buffer común junto con el código de que se trata del último paquete, para que el resto de hilos estén informados. Por último se liberan todos los recursos y termina la ejecución del hilo.

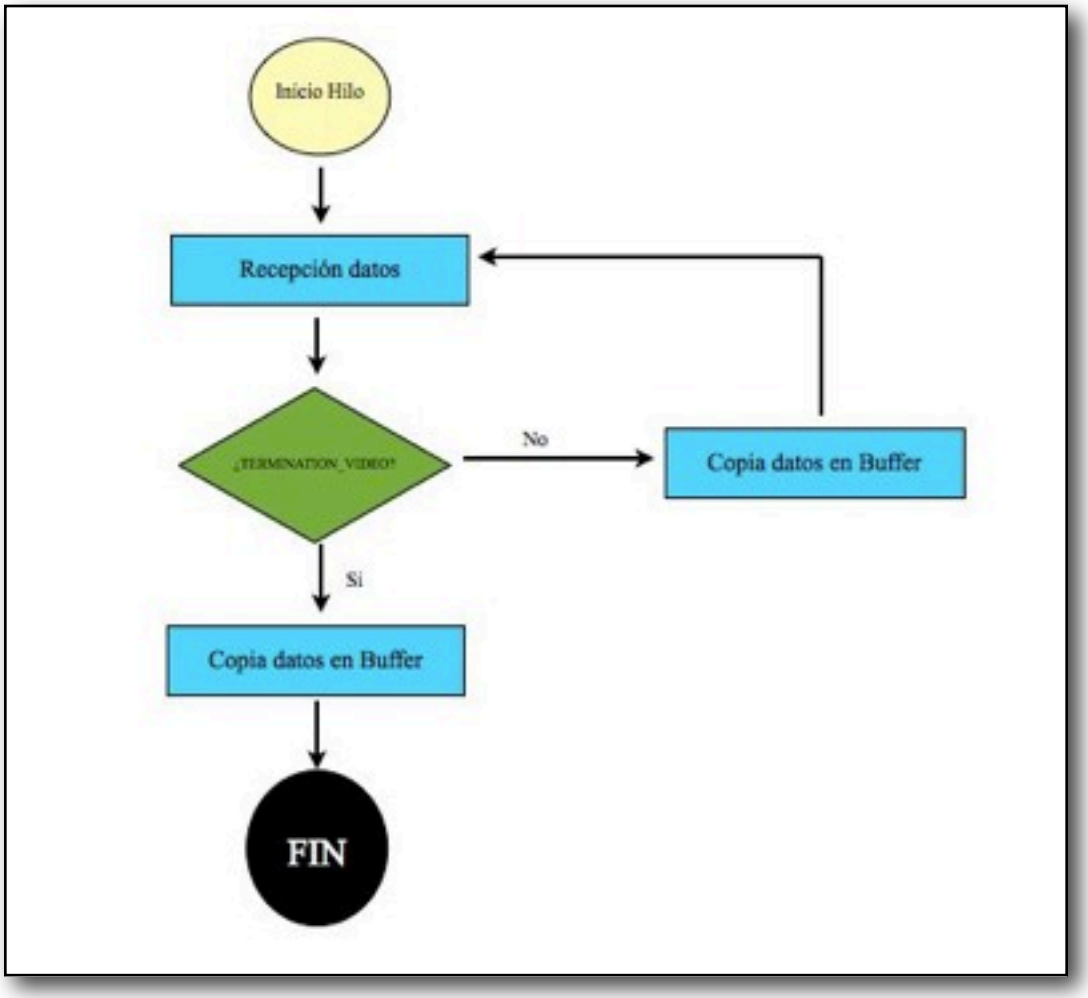


Ilustración 34. Diagrama de estados del hilo Receptor

Insertor Datos del Vídeo en la Fila

Este hilo (*insercionFila*) se encuentra entre medias del hilo Receptor y el hilo Reproductor. Su cometido es mantenerse a la espera de que el hilo Receptor haya obtenido los datos suficientes para copiarlos en el Buffer Común que comparten. Una vez copiado, el Receptor activa un *flag* que se encuentra en permanente escucha por este hilo, y es en ese momento en el que se introduce el paquete (Buffer Común) dentro de la Fila gracias a la función "*insertar*", de la que posteriormente intentará obtener los paquetes el hilo Reproductor. Como se muestra en la *Ilustración 35*, este mecanismo se repite hasta que se recibe el último paquete de datos.

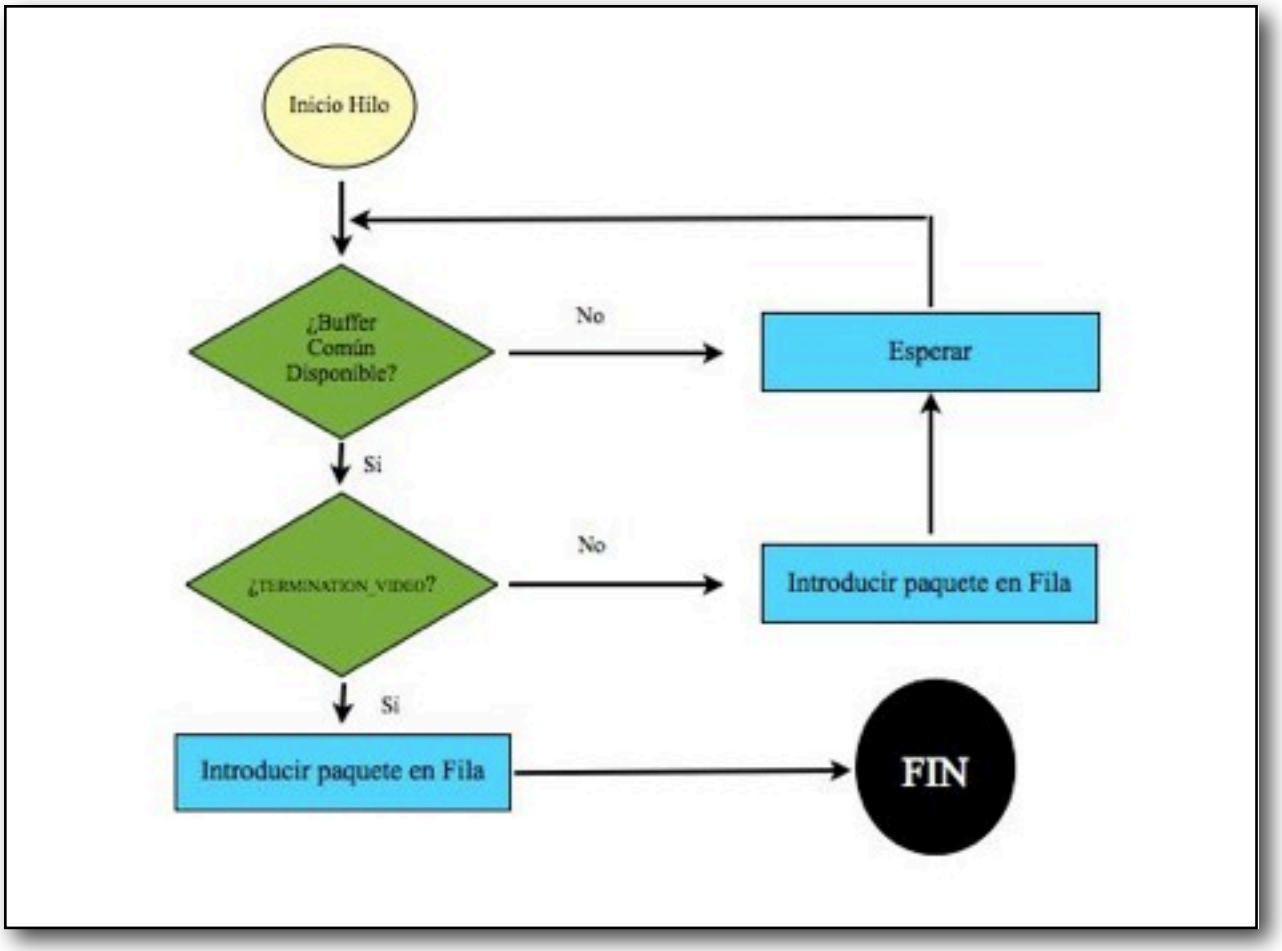


Ilustración 35. Diagrama de estados del hilo Insertor

Esta función inserta dentro de la estructura de la fila, los paquetes con los datos del vídeo en orden de llegada. Esta función se encuentra sincronizada con con la función "*obtener*", mediante una lógica de exclusión mutua de los hilos evitando así el uso simultáneo de recursos comunes, para mantenerla informada del número de paquetes que hay en la "*Fila"*, y evitar eliminar uno cuando la lista se encuentra vacía. En el caso en el que no existan más paquetes porque la transmisión del vídeo ha finalizado, envía el código de finalización para terminar la visualización y reproducción.

Reproducción del Vídeo

Este hilo (*mostrarVideo*) es el encargado de iniciar la reproducción de los datos que son obtenidos por la aplicación. El proceso se mantiene a la espera de que haya un número mínimo de paquetes en la fila para conseguir que la posterior visualización sea continua y sin cortes.

Una vez conseguido ese mínimo, se crea la estructura *VideoState* que contiene todos los datos que son necesarios para la reproducción, además, esta estructura es introducida como parámetro a las funciones que realizan la visualización ya que mantiene el estado de los datos.

Otra de las acciones que realiza este hilo es el registro de todos los formatos y *códecs* que se encuentran dentro de la herramienta *ffmpeg* para poder decodificar cualquier vídeo que sea transmitido, la inicialización de la ventana en la que posteriormente se visualizará el vídeo y por último la creación del hilo encargado de la visualización (*decode_thread*). El proceso se mantiene activo a la espera de eventos que son enviados por el visualizador y únicamente finaliza cuando obtiene el evento que cierra la ventana de visualización o cuando se han terminado la reproducción del vídeo como se muestra en la *Ilustración 36*.

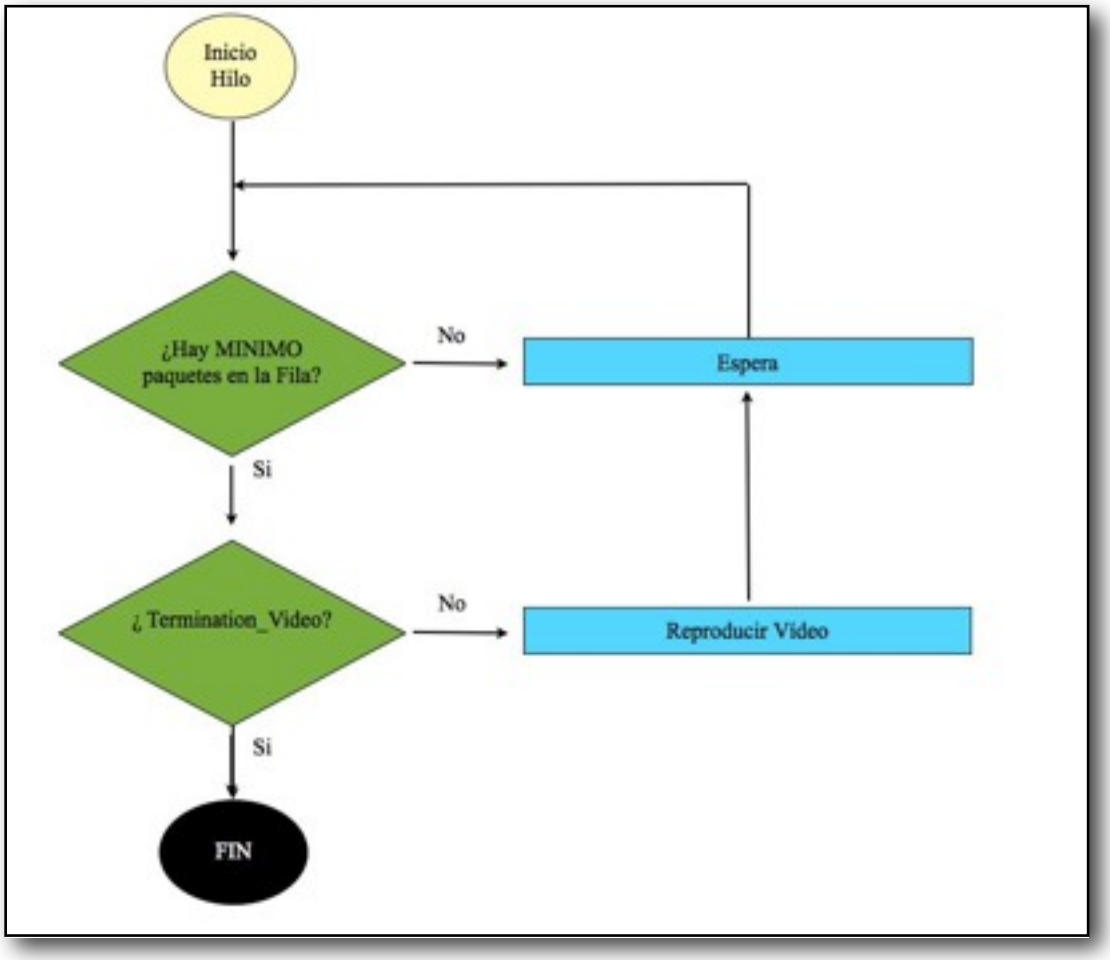


Ilustración 36. Diagrama de estados del hilo Reproductor

Visualización del Vídeo

Este hilo (*decode_thread*) es el encargado de la decodificación y visualización de los datos mediante la llamada a diversos métodos que realizan esas funciones y gracias a la estructura común que contiene los datos y el estado del vídeo (*VideoState*). Para ello, lo primero se crea una estructura (*sd_video*) en la que se van a copiar los datos que se van a ir obteniendo de forma síncrona mediante la función "*obtener*". La *Ilustración 37* muestra los estados del hilo de visualización.

Esta función devuelve, de la estructura de la fila, los datos de los paquetes que contiene, en el mismo orden en el que han sido introducidos, para ello, como se dijo en el *hilo Insertor*, mantiene un estado continuo de sincronización con la función "insertar", por la cual, únicamente obtiene paquetes mientras exista alguno dentro de la lista, en el caso de que no haya, se mantiene a la espera de la inserción de alguno. En el caso en el que no existan más paquetes porque la transmisión del vídeo ha finalizado, envía el código de finalización al hilo que lo llama para la terminar la visualización y reproducción del vídeo.

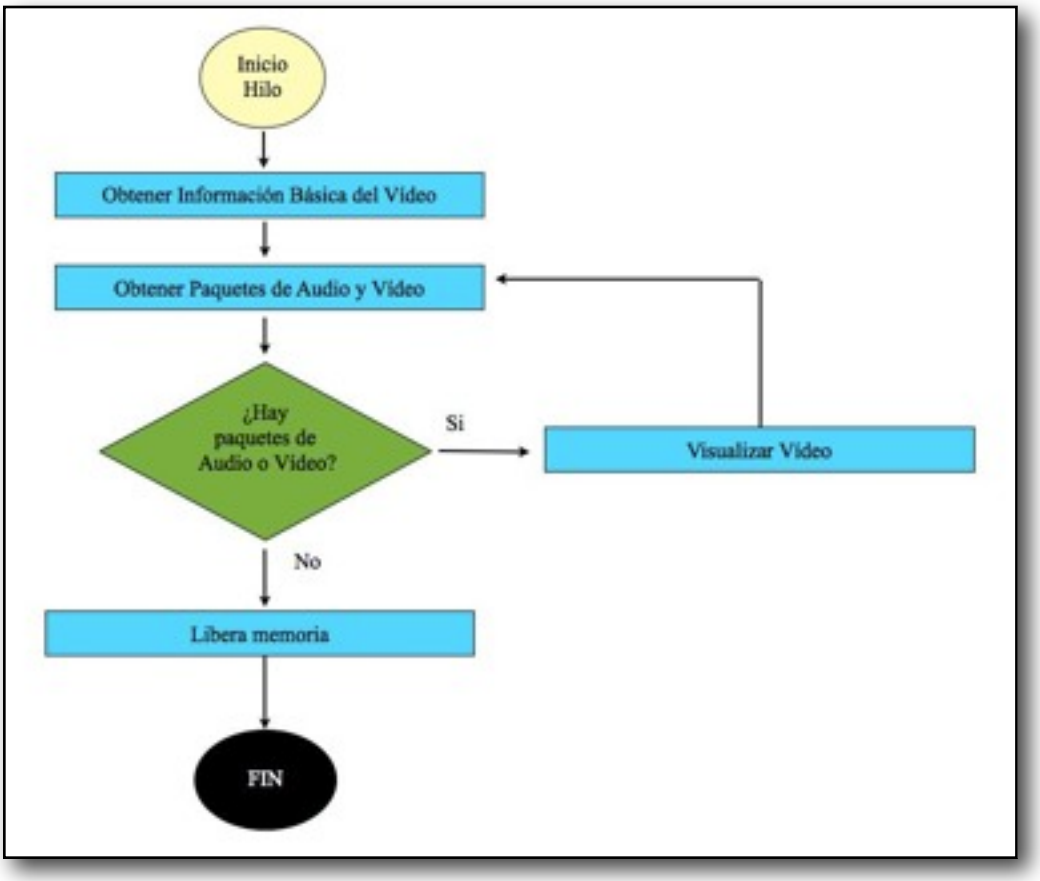


Ilustración 37. Diagrama de estados del hilo Visualizador

La estructura creada, a su vez, es la encargada de crear un objeto que contiene los *bytes* gracias a los cuales se obtienen los flujos de datos necesarios para iniciar la decodificación. Para la correcta visualización, se deben de obtener un número mínimo de flujos que contienen la información fundamental del vídeo: formato, *códec* y otros datos necesarios para iniciar el proceso de visualización.

El siguiente paso es la obtención del número de flujos de vídeo y de audio que contiene la película para abrir los componentes necesarios para decodificar los flujos, para ello se utiliza la función "*stream_component_open*".

Esta función es la encargada de encontrar el tipo de códec necesario para el flujo que recibe. En el caso de audio, inicializa los valores fundamentales como la frecuencia, el formato, los canales, el decodificador, etc; y crea una lista en la que se introducirán los paquetes de audio. Además, realiza una llamada a una función (audio_callback) que es la encargada de obtener los paquetes de audio de la lista, decodificarlos, sincronizarlos y reproducirlos.

En el caso de flujos de vídeo, al igual que en los de audio, además de iniciar otra lista diferente en la que se encontrarán los paquetes de vídeo, crea un hilo (video_thread) encargado de obtener los paquetes de vídeo que se encuentran dentro de la lista, decodificarlos, sincronizar las imágenes y mostrarlas por la pantalla.

Hay que destacar, que las listas de paquetes de audio y de vídeo, están sincronizadas, con las funciones que las llaman, es decir, únicamente se pueden obtener paquetes en el caso de que existan, en otro caso, se mantienen a la espera.

Mientras se están realizando las acciones anteriores, el hilo (*decode_thread*) se mantiene en una espera activa en la cual está obteniendo paquetes de la estructura que contiene los datos (*sd_video*) y a su vez, introduce cada paquete en su correspondiente lista: audio o vídeo. Una vez terminada la visualización del vídeo la ventana de visualización se cierra y todos los hilos que realizaban las tareas de visualización, reproducción y recepción terminan sus ejecuciones. En caso de cualquier fallo durante la reproducción o cierre de la ventana del vídeo por parte del usuario, la visualización termina.

Abandono DDS

La aplicación no termina aunque la visualización del vídeo haya concluido, se mantienen activos los hilos publicadores y subscriptores de los mensajes de texto. Una vez el usuario decida terminar con el "*Chat*", es cuando se realiza la liberación de los recursos de los hilos y estructuras que han sido creadas, así como el abandono del dominio de participación *DDS* creado al inicio de forma similar a cómo se explicó en el *Paso 3 de la Aplicación Inicial del Capítulo 3*, pero con los elementos creados en ésta.

4.5 Conclusiones

Tras la finalización del desarrollo se puede concluir que se ha conseguido realizar la aplicación como se deseaba, no sin un esfuerzo especial a la hora de la recepción de los datos y del control de los parámetros de calidad de servicio que hicieran posible un correcto funcionamiento de la aplicación para los diferentes tipos de visualización que posee.

Además, a lo largo del desarrollo de la aplicación se han podido realizar varias mejoras que posibilitaban un mejor funcionamiento de la misma, las cuales no se encontraban especificadas al inicio del proyecto aunque se han especificado como requisitos en el *apartado 4.3*.

A destacar la configuración inicial que realiza el usuario tanto en la aplicación transmisora como de la aplicación receptora, el cual tiene la posibilidad de decidir en que tópicos escribe y recibe los mensajes, así como la capacidad de decidir en qué tópico envía el vídeo (Transmisor) o de qué tópico recibe el vídeo (Receptor).

Además, existe la posibilidad añadida de poder seleccionar, por parte del receptor, la forma de visualización del vídeo. En un principio la aplicación, únicamente admitía la visualización del vídeo en directo. En una posterior mejora, se añadió la posibilidad de recibir el vídeo completo, aunque el envío en vivo hubiese finalizado.

Una última mejora que se realizó, es la posibilidad de guardar el vídeo que es recibido si el usuario receptor así lo deseaba. Para ello, se debe introducir una ruta y un nombre del fichero a guardar dentro de la configuración inicial del Receptor.

En definitiva, el desarrollo de la aplicación se ha convertido en un reto, debido a la complejidad interna que posee FFmpeg, así como por la multitud de posibilidades de diseño de un entorno DDS, a causa de la falta de documentación existente fuera de las aplicaciones de ejemplo que poseen. Aún así, el *Capítulo 3* de iniciación a las tecnologías y herramientas, ayudó a la consecución de este desarrollo final.

Capítulo 5:
Pruebas del sistema

"Lo bueno necesita aportar pruebas; lo bello, no".

Bernard Le Bouvier de Fontenelle *(1657-1757) Escritor Francés*

En este capítulo se realizan las pruebas sobre la aplicación desarrollada. Para ellos se pasan una serie de test que son significativos a la hora de evaluar el correcto funcionamiento de la aplicación. Finalmente se exponen unas conclusiones sobre los datos obtenidos.

5.1 Introducción

En este apartado se va a evaluar el funcionamiento de la aplicación desarrollada. El entorno en el que se han realizado las pruebas, como se muestra en la *Ilustración 38*, ha consistido en dos máquinas que se encuentran conectadas mediante una red *Ethernet.* La primera de las máquinas en una máquina virtual con sistema operativo Linux montada sobre un portátil MacBook Pro, en ella se iniciará la aplicación transmisora. La otra máquina es un PC de sobremesa con un sistema Linux en donde se iniciará la aplicación receptora.

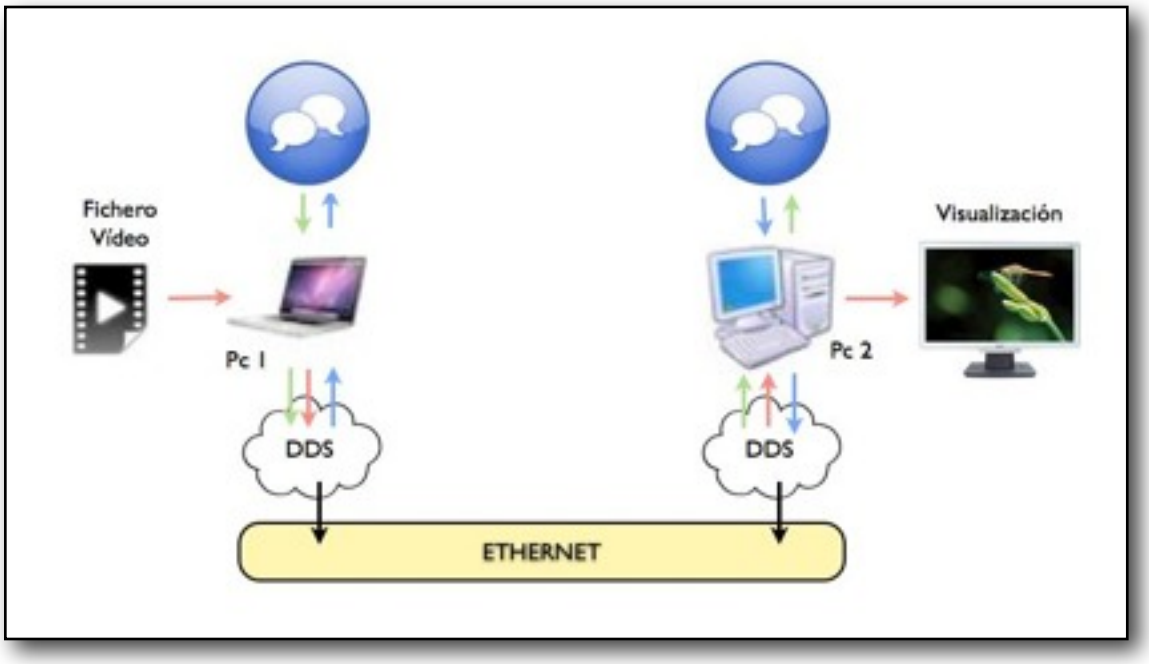


Ilustración 38. Entorno de pruebas

Las pruebas que se han realizado consisten en el correcto funcionamiento de la aplicación, para ello se han contemplado pruebas que verifiquen el adecuado envío y recepción de los datos realizando unas medidas de los tiempos que tarda la aplicación en realizar su cometido, así como la correcta visualización de los datos transmitidos en función del número de usuarios y formatos de vídeo enviados. Finalmente se redactan las conclusiones a las que se han llegado tras las pruebas.

5.2 Pruebas de Recepción

En esta prueba se intenta verificar la correcta recepción de los datos para los dos tipos de recepción que existen: Live o VoD. Para ello se modifica el orden y el tiempo de inicio de cada una de las aplicaciones (Transmisor y Receptor).

- Modo Live

Este modo consiste en la obtención de los datos por parte del Receptor en directo, es decir, únicamente se reciben los datos que están siendo transmitidos en ese momento.

Para esta prueba el sistema OpenSplice se mantiene activo siempre en el nodo transmisor y se varía su inicio en recepción. Los casos más conflictivos son cuando el nodo receptor inicia OpenSplice al mismo tiempo que el emisor y cuando lo inicia mucho más tarde.

También se varía el momento en el que se inicia la aplicación dentro de cada nodo pudiendo observar cual es su comportamiento.

	Inicio OpenSplice Transmisor = Receptor	Inicio OpenSplice Transmisor antes Receptor
Inicio Aplicación Receptor antes Transmisor	Recibe todos los datos	No se reciben los datos
Inicio Aplicación Receptor mitad Transmisor	Recibe los datos a partir del momento en el que se inicia el Receptor	No se reciben los datos
Inicio Aplicación Receptor después Transmisor	No se reciben los datos	No se reciben los datos

Tabla 8. Pruebas modo Live

Como se puede observar en la *Tabla 8*, en modo Live únicamente se reciben los datos cuando OpenSplice está iniciado antes o al mismo tiempo en el nodo Receptor y en el nodo Transmisor.

Además la aplicación Receptora debe de ser iniciada antes que la Transmisora para poder recibir todos los datos, ya que si se inicia ligeramente después que la Transmisora no se reciben todos los datos, sólo se reciben los que se obtienen a partir de ese momento. En cualquier otro caso, la aplicación Receptora no recibe ningún dato cuando se inicia en modo Live.

- Modo VoD

Este modo consiste en la obtención de los datos por parte del Receptor en cualquier momento, es decir, el usuario recibe todos los datos aunque la transmisión de los mismos haya finalizado.

Como en el caso anterior, se varía el momento de iniciación de OpenSplice en cada uno de los nodos, así como de la aplicación para poder observar el comportamiento que tiene.

	Inicio OpenSplice Transmisor = Receptor	Inicio OpenSplice Transmisor antes Receptor
Inicio Aplicación Receptor antes Transmisor	Recibe todos los datos	Recibe todos los datos
Inicio Aplicación Receptor mitad Transmisor	Recibe todos los datos	Recibe todos los datos
Inicio Aplicación Receptor después Transmisor	Recibe todos los datos	Recibe todos los datos

Tabla 9. Pruebas modo VoD

Como se puede observar en la *Tabla 9*, para este modo de recepción los datos transmitidos son recibidos en cualquier caso. Esto es debido a que el sistema está configurado para que se mantengan los datos en él, y en el momento que un usuario solicite esos datos, sean enviados todos en ese momento.

OpenSplice, por defecto, únicamente mantiene una muestra (dato publicado) en el sistema. Para modificar este comportamiento se debe de modificar el parámetro de calidad de servicio *History*, que como se comentó en la *Tabla 1*, especifica cuántos datos serán almacenados en la infraestructura DDS.

Si se publican datos y se llega al número especificado por este parámetro sin que el lector las haya obtenido, el escritor, sobreescribe esos datos con nuevos en el sistema.

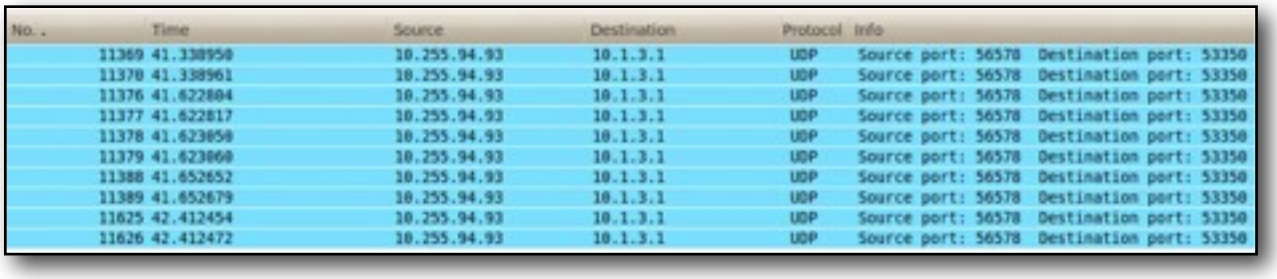
Podemos concluir que la recepción de los datos es correcta cuando se trata de un entorno Ethernet y dependiendo del modo en el que se inicie la aplicación se obtendrán unos resultados u otros. Con el modo Live se conseguirá únicamente la visualización del vídeo en el momento que se conecta y los mensajes de Chat enviados a partir de ese momento (más el último mensaje que fue enviado al sistema), mientras que en el modo VoD se conseguirá visualizar el vídeo completo independientemente del momento en que se inicie la aplicación, así como todos los mensajes de texto enviados al entorno.

5.3 Pruebas de Tiempo

Las pruebas de tiempo se han realizado calculando los tiempos empleados en la transmisión del vídeo completa y el tiempo de envío entre paquetes. Para ello se ha utilizado la herramienta *WireShark* que realiza un rastreo de los paquetes enviados a la red.

- Tiempo envío total

Para el tiempo de envío total se han realizado 5 medidas, observando el momento en el que se envía el primer paquete y el último, para calcular posteriormente su media y su varianza.



No.	Time	Source	Destination	Protocol	Info
11369	41.338950	10.255.94.93	10.1.3.1	UDP	Source port: 56578 Destination port: 53350
11370	41.338961	10.255.94.93	10.1.3.1	UDP	Source port: 56578 Destination port: 53350
11376	41.622804	10.255.94.93	10.1.3.1	UDP	Source port: 56578 Destination port: 53350
11377	41.622817	10.255.94.93	10.1.3.1	UDP	Source port: 56578 Destination port: 53350
11378	41.623050	10.255.94.93	10.1.3.1	UDP	Source port: 56578 Destination port: 53350
11379	41.623060	10.255.94.93	10.1.3.1	UDP	Source port: 56578 Destination port: 53350
11388	41.652652	10.255.94.93	10.1.3.1	UDP	Source port: 56578 Destination port: 53350
11389	41.652679	10.255.94.93	10.1.3.1	UDP	Source port: 56578 Destination port: 53350
11625	42.412454	10.255.94.93	10.1.3.1	UDP	Source port: 56578 Destination port: 53350
11626	42.412472	10.255.94.93	10.1.3.1	UDP	Source port: 56578 Destination port: 53350

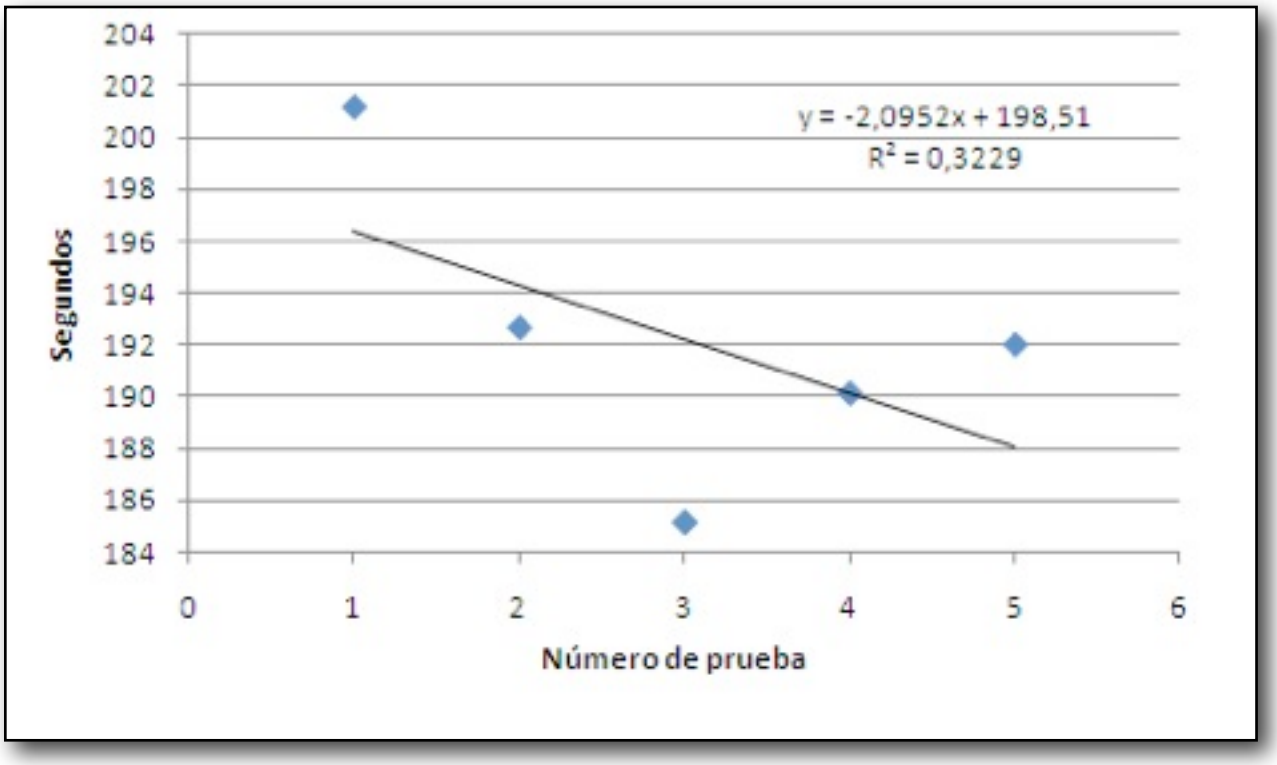
Tabla 10. Tiempos de transmisión con WireShark

Como se observa en la *Tabla 10*, el primer paquete que se transmite corresponde al número 11369 de los que ha estado escuchando el programa. El instante en el que se ha capturado es en el segundo 41.338950, cuyo emisor corresponde a la IP 10.255.94.93 y el destino es la IP 10.1.3.1. Se trata de un paquete UDP enviado a través del puerto 56578 y recibido en el puerto 53350 que es el que tiene OpenSplice reservado para la recepción de datos.

	Tiempo Inicio (seg)	Tiempo Final (seg)	Tiempo TOTAL (seg/min)
Prueba 1	14,82336100	216,03379900	201,210438 / 3,3535
Prueba 2	18,78442500	211,44646500	192,66204 / 3,2110
Prueba 3	35,46236400	220,57850300	185,116139 / 3,0852
Prueba 4	21,23471200	211,35426700	190,119518 / 3,1686
Prueba 5	33,76934200	225,77523400	192,005888 / 3,2000

Tabla 11. Tiempos Inicio-Fin

Obteniendo los tiempos de la *Tabla 11*, tras la realización de 5 transmisiones completas del vídeo obtenemos las siguiente gráfica de dispersión de los resultados.



Gráfica 1. Dispersión del tiempo total

Como se observa en la *Gráfica 1*, la línea de tendencia de los datos es descendente teniendo como varianza **0,3229** y una media de envío total de los datos de **192,2228046 segundos.**

- Tiempo de envío entre paquetes

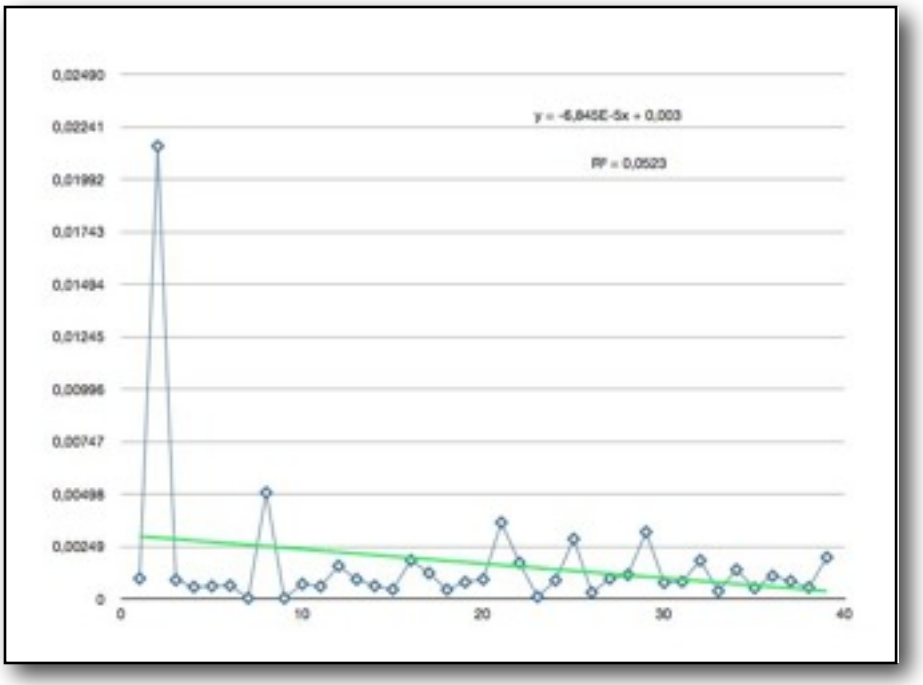
El tiempo de envío entre paquetes se ha obtenido tras la realización de tres pruebas en las que se han recogido los datos de cuarenta paquetes por cada prueba. En la *Tabla 12* se muestra el resultado de la diferencia entre los tiempos de llegada de cada paquete con el anterior en cada una de las pruebas.

	Prueba 1	Prueba 2	Prueba 3
Paquete 2 - Paquete 1	0,00096100	0,00134300	0,00118100
Paquete 3 - Paquete 2	0,02148800	0,00059600	0,00042600
Paquete 4 - Paquete 3	0,00089400	0,00089500	0,00052300
Paquete 5 - Paquete 4	0,00055900	0,00069100	0,00064900
Paquete 6 - Paquete 5	0,00059800	0,00065400	0,00001800
Paquete 7 - Paquete 6	0,00063000	0,00077100	0,00500400
Paquete 8 - Paquete 7	0,00002500	0,00002500	0,00003300
Paquete 9 - Paquete 8	0,00504000	0,00523200	0,00525800
Paquete 10 - Paquete 9	0,00002400	0,00003700	0,00002100
Paquete 11 - Paquete 10	0,00070000	0,00013900	0,00056600
Paquete 12 - Paquete 11	0,00059000	0,00013800	0,00065500
Paquete 13 - Paquete 12	0,00156000	0,00082700	0,00034900
Paquete 14 - Paquete 13	0,00092000	0,00165900	0,00088500
Paquete 15 - Paquete 14	0,00061000	0,00023800	0,00081000
Paquete 16 - Paquete 15	0,00044000	0,00075500	0,00054300
Paquete 17 - Paquete 16	0,00183000	0,00090600	0,00076000
Paquete 18 - Paquete 17	0,00122000	0,00099300	0,00163100
Paquete 19 - Paquete 18	0,00043000	0,00156500	0,00030700
Paquete 20 - Paquete 19	0,00079000	0,00094000	0,00055300
Paquete 21 - Paquete 20	0,00092000	0,00013000	0,00176000
Paquete 22 - Paquete 21	0,00362000	0,00006200	0,00127500
Paquete 23 - Paquete 22	0,00171000	0,00033700	0,00126000
Paquete 24 - Paquete 23	0,00009000	0,00063800	0,00022200
Paquete 25 - Paquete 24	0,00088000	0,00105300	0,00092100

Paquete 26 - Paquete 25	0,00284000	0,00084700	0,00106300
Paquete 27 - Paquete 26	0,00029700	0,00102300	0,00209100
Paquete 28 - Paquete 27	0,00095300	0,00081400	0,00124500
Paquete 29 - Paquete 28	0,00114000	0,00065600	0,00112200
Paquete 30 - Paquete 29	0,00317000	0,00164500	0,00046200
Paquete 31 - Paquete 30	0,00076000	0,00093500	0,00050400
Paquete 32 - Paquete 31	0,00081000	0,00078300	0,00083700
Paquete 33 - Paquete 32	0,00182000	0,00090600	0,00174500
Paquete 34 - Paquete 33	0,00036100	0,00051700	0,00087900
Paquete 35 - Paquete 34	0,00138200	0,00174000	0,00067300
Paquete 36 - Paquete 35	0,00050400	0,00055600	0,00111800
Paquete 37 - Paquete 36	0,00108100	0,00077600	0,00033500
Paquete 38 - Paquete 37	0,00084100	0,00062600	0,00071300
Paquete 39 - Paquete 38	0,00053500	0,00111700	0,00056800
Paquete 40 - Paquete 39	0,00196500	0,00030700	0,00154100

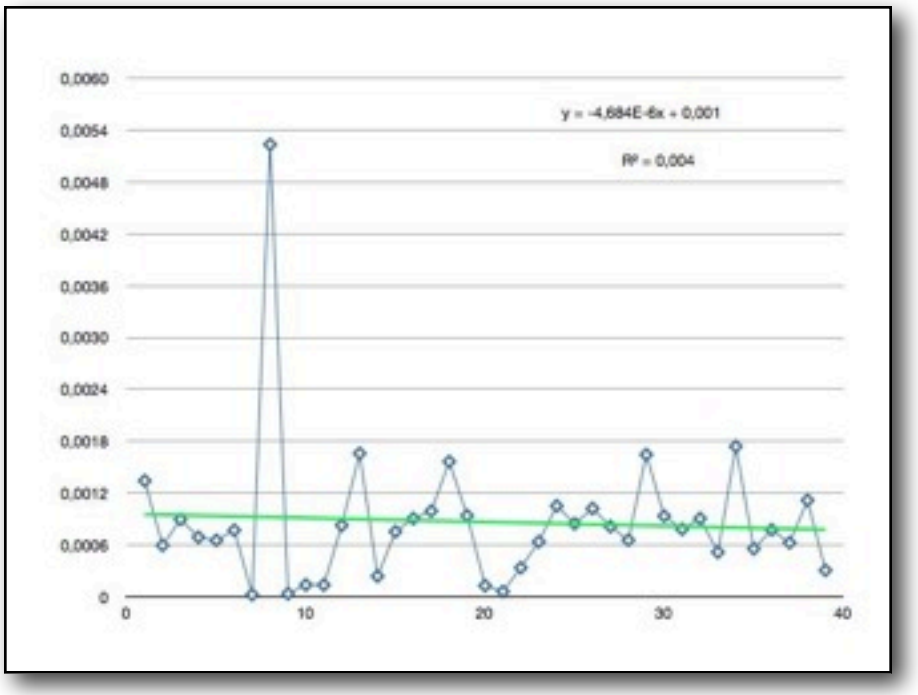
Tabla 12. Tiempo entre paquetes

Tras la toma de las medidas se han realizado las gráficas de dispersión de la diferencia entre paquetes, correspondientes a cada prueba, obteniendo los siguientes resultados.



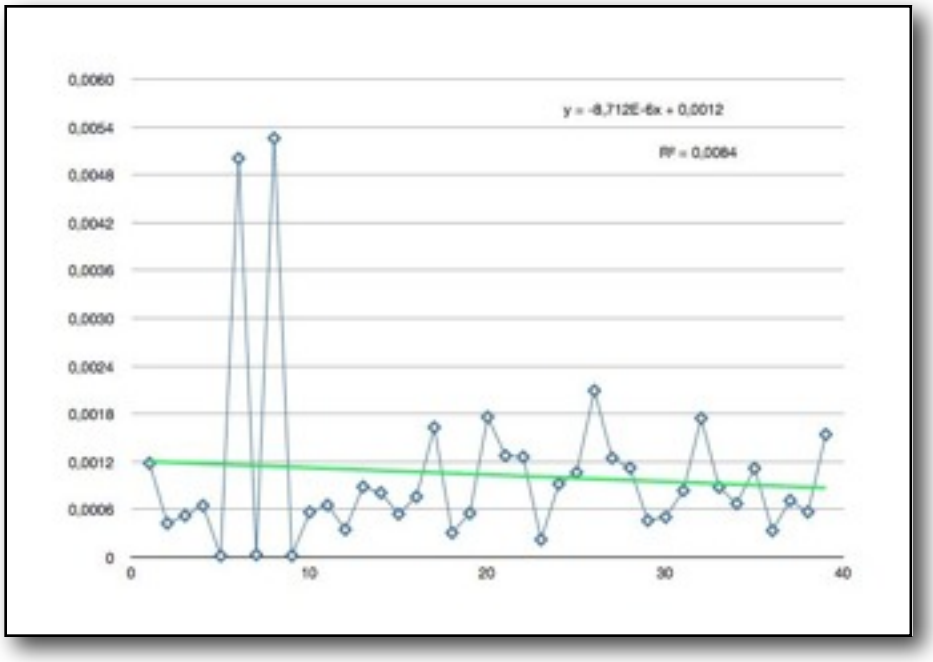
Gráfica 2. Dispersión entre paquetes de la prueba 1

Como se observa en la *Gráfica 2*, entre el paquete 3 y 2 se obtiene una gran diferencia que poco a poco se va estabilizando en los siguientes. La línea de tendencia de los datos es claramente descendente.



Gráfica 3. Dispersión entre paquetes de la prueba 2

Como se observa en la *Gráfica 3*, en términos generales se obtienen unos valores muy estables a lo largo de la prueba, excepto entre los paquetes 9 y 8 en el que se produce un retardo más significativo. La línea de tendencia de los datos es muy constante aún siendo ligeramente descendente.



Gráfica 4. Dispersión entre paquetes de la prueba 3

Como se observa en la *Gráfica 4*, los valores que se obtienen son más dispersos dentro de la poca diferencia que existe entre ellos. La línea de tendencia es ligeramente descendente.

Tras la realización de las pruebas, en la *Tabla 13*, se muestran los valores más representativos de las mismas, siendo la prueba 2 en la que mejores datos se han obtenido a nivel global.

	Máximo	Mínimo	Media	Varianza
Prueba 1	0,02148800	0,00002400	0,00166636	0,05230000
Prueba 2	0,00523200	0,00002500	0,00086853	0,00400000
Prueba 3	0,00525800	0,00001800	0,00103861	0,00840000

Tabla 13. Valores representativos de las pruebas

Tras la finalización de las pruebas de tiempo sobre la aplicación el análisis que se obtiene es que la aplicación responde correctamente al envío de los datos. El reducido tiempo que se obtiene en el envío entre paquetes se traduce en un tiempo total de envío de vídeo que hace posible una reproducción continua, por lo que apenas se producen cortes en la visualización en el entorno en el que se han desarrollado las pruebas.

5.4 Pruebas de Formato

Las pruebas de visualización consisten en verificar que los vídeos que se están transmitiendo se visualizan correctamente en el Receptor. Para ello se han realizado pruebas enviando diferentes formatos de vídeo.

- Mpeg

En esta prueba se realiza la transmisión de un vídeo en formato Mpeg-ts, para su visualización en el nodo receptor.

Ilustración 39. Imagen vídeo Mpeg-ts [25]

Como se muestra en la *Ilustración 39*, el vídeo se visualiza correctamente. La recepción de los datos es correcta y no se producen cortes en la reproducción. Tanto la imagen como el sonido se observa y escucha de forma nítida.

- Avi

En esta prueba se realiza la transmisión de un vídeo en formato Avi, para su visualización en el nodo receptor.

Ilustración 40. Imagen vídeo Avi [26]

En este caso, como se muestra en la *Ilustración 40*, el vídeo no se visualiza correctamente, se produce una gran pixelación de las imágenes. Esto puede ser producido, porque la decodificación de los frames, por parte de la herramienta FFmpeg, no sea correcta.

- Otro formato

Cuando la aplicación Transmisora envía un vídeo en un formato que no se corresponde a los anteriores, aunque los datos son recibidos en la aplicación Receptora, la visualización del vídeo no se inicia, ya que se trata de un formato que no es soportado por la misma. Aún así, el sistema de Chat sigue funcionando, por lo que no impide la transmisión y recepción de los mensajes de texto enviados.

Tras la realización de las pruebas de formato, el análisis que se obtiene es que para formato de tipo Mpeg-ts la visualización se realiza correctamente, sin apenas retardo ni pixelación en la imagen; para la visualización en formato Avi, se deben de realizar mejoras, ya que no se consigue una reproducción correcta del vídeo; y para otros formatos, la aplicación responde correctamente, permitiendo la comunicación por chat, pero sin visualizar el vídeo debido a que no están soportados.

5.5 Pruebas de Visualización

En esta prueba se han realizado diferentes comprobaciones de visualización con varios transmisores y receptores para observar el comportamiento de la aplicación:

- 1 Transmisor y 2 Receptores
- 2 Transmisores y 1 Receptor
- 2 Transmisores y 2 Receptores

1. Envío de vídeo. 1 Transmisor y 2 Receptores

En este caso se ha utilizado un usuario que publica un vídeo sobre el tópico “Vídeo” con otros dos usuarios que están subscritos al mismo. Para ello se han dado de alta en el tópico en el que está siendo publicado dicho vídeo.

Ilustración 41. Vídeo 1 Transmisor y 2 Receptores [25]

Como se observa en la *Ilustración 41*, el vídeo es visualizado por ambos usuarios Receptores. En este caso no existe apenas diferencia entre un vídeo y otro debido a que los usuarios han iniciado la aplicación en el mismo instante.

2. Envío de vídeo. 2 Transmisores y 1 Receptor

En la siguiente prueba se van a utilizar dos usuarios transmisores, los cuales van a publicar los datos sobre el mismo tópico “Video”. En este caso únicamente un Receptor estará subscrito a dicho tópico.

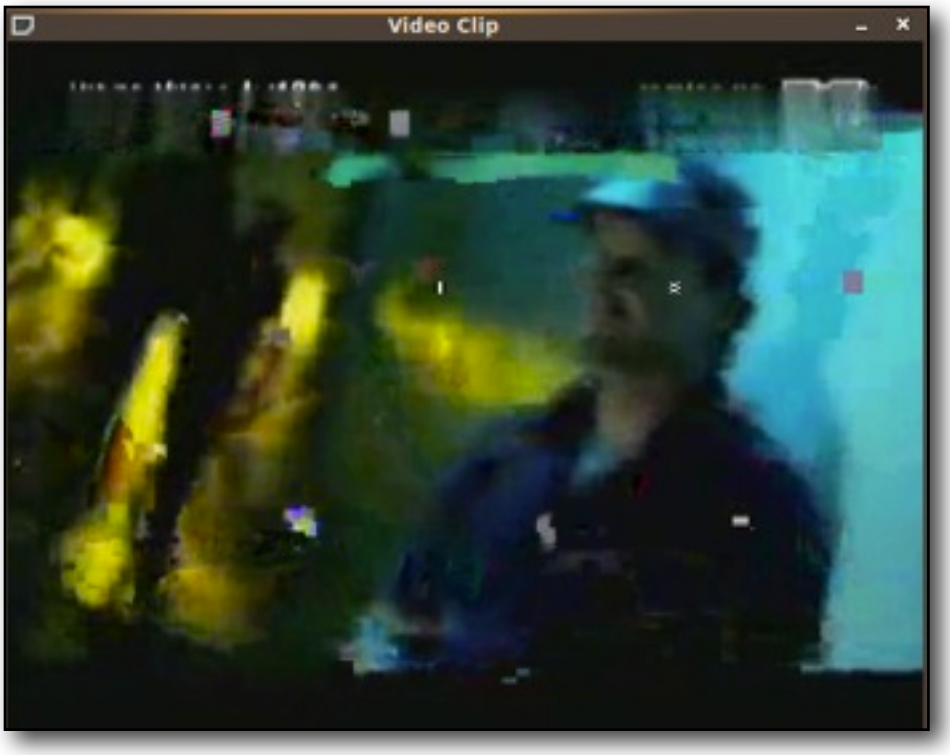


Ilustración 42. Vídeo 2 Transmisores y 1 Receptor [25]

Como se observa en la *Ilustración 42*, en este caso se produce una mala visualización del vídeo. Esto es debido a que ambos transmisores están publicando datos de diferentes vídeos sobre un mismo tópico, lo que produce el solapamiento de los datos en el receptor. Se trata de un caso extremo, ya que lo normal es que la transmisión de vídeos diferentes se realice en diferentes tópicos.

3. Envío de vídeo. 2 Transmisores y 2 Receptores

Para esta última prueba se van a utilizar dos transmisores publicando dos vídeos sobre dos tópicos diferentes “Video1” y “Video2”. A cada tópico van a estar subscritos dos receptores que visualizarán los datos del vídeo que se encuentren en cada tópico.

Ilustración 43. Vídeo 2 Transmisores y 2 Receptores [25]

Como se observa en la *Ilustración 43*, los usuarios receptores están visualizando los vídeos independientemente. Cada vídeo, es el publicado en cada uno de los tópicos. En este caso la visualización es correcta en ambos receptores.

El análisis que se puede obtener de estas pruebas es que la aplicación responde en casi todos los casos en los que hay comunicación entre diferentes usuarios. El único momento en el que se produce una situación anómala, es cuando se intenta enviar sobre el mismo tópico dos vídeos diferentes. En el resto de los casos, la aplicación responde correctamente tanto para vídeo en modo Live como en modo VoD para uno o múltiples usuarios.

5.6 Requisitos cumplidos

En este apartado se va a mostrar una tabla con los requisitos que se han conseguido lograr tras la realización de las pruebas, dentro de los que se redactaron en el *apartado 4.3*. Los códigos de verificación de la *Tabla 14* indican si la aplicación cumple el requisito (X), si el requisito viene intrínseco por la utilización de DDS (O), o si el requisito se cumple parcialmente (+/-).

	Prueba 1 Recepción	Prueba 2 Tiempo	Prueba 3 Formato	Prueba 4 Visualización
RF1	X	X		X
RF2				X
RF3	O	O	O	O
RF4	X			
RF5	X			
RF6	O	O	O	O
RF7			X	X
RF8	X			
RF9	X	X		X
RF10	X			
RNF1	O	O	O	O
RNF2	O	O	O	O
RNF3	O	O	O	O
RNF4	O	O	O	O
RNF5	O	O	O	O
RNF6	O	O	O	O
RNF7	O	O	O	O
RNF8	O	O	O	O
RNF9	O	O	O	O
RNF10	O	O	O	O
RNF11	O	O	O	O
RNF12	X			X
RNF13			+/-	+/-

RNF14			+/-	+/-
RNF15	+/-	+/-	+/-	+/-
RNF16	X	X		X
RNF17	X			
RNF18	X			
RNF19	X	X		
RNF20	X			
RNF21	X			
RNF22	X			X
RNF23	X			X
RNF24	X			X

Tabla 14. Tabla de resultado sobre los requisitos

Como se puede observar, la gran mayoría de los requisitos son cumplidos por la aplicación, ya sea por el tipo de tecnología que está utilizando, como es DDS, que incorpora la mayoría de los requisitos en sus características, o por la propia aplicación que tras la realización de las pruebas se ha comprobado que consigue su cometido.

La mayor parte de los requisitos consistían en que tanto la recepción de los datos, como su visualización se realizase de forma correcta y en el menor tiempo posible, de ahí que la mayoría de los requisitos se cumplan en estas pruebas.

Existen algunos requisitos, los correspondientes a las pruebas de formato y visualización, que se cumplen de forma parcial, los cuales están propuestos como futuras mejoras en el *apartado 6.2*. Además el requisito RNF15, correspondiente al tipo de dispositivo en el que se puede utilizar, no llega a cumplirse en dispositivos móviles. En el *apéndice A.5* se explica el problema.

5.7 Conclusiones

Tras la realización de las pruebas se observa que la aplicación contiene algunas deficiencias en cuanto a los formatos que permite visualizar y quizás en la gestión de los tópicos a la hora de una publicación de datos de vídeo sobre un mismo tópico, pero en términos generales funciona correctamente.

Por otro lado, el envío y recepción de datos se realiza en un tiempo aceptable, gracias a lo cual se puede realizar una correcta visualización de los vídeos tanto en modo VoD como en modo Live. Además, gracias a los pruebas realizadas, se han obtenido resultados que posibilitan unas mejoras futuras que se redactan en el *Capítulo 6.2*.

Finalmente, aunque las pruebas se han realizado dentro de un entorno relativamente favorable, como es una red Ethernet, en la que apenas se producen pérdidas o errores en la comunicación, se puede concluir con que la aplicación cumple su cometido que es la transmisión de vídeo bajo un sistema distribuido de datos y su correcta visualización en diferentes nodos.

Capítulo 6: Conclusiones y líneas futuras

"Conclusión es el lugar donde llegaste cansado de pensar".

Anónimo

En este capítulo se redactan las conclusiones a las que se han llegado tras la realización de la aplicación y las posteriores líneas de trabajo que se pueden continuar.

6.1 Conclusiones

Tras la finalización del proyecto de fin de carrera se ha conseguido llevar a cabo el estudio y desarrollo de una tecnología de *middleware* de Publicación/ Suscripción basada en un estándar oficial del *OMG* implementada por la empresa *PrimsTech*, la cual está orientada al desarrollo de aplicaciones críticas de tiempo real.

Se ha realizado una investigación sobre su funcionamiento, haciendo especial hincapié tanto es sus parámetros de calidad de servicio, como en su arquitectura (publicador/subscriptor). Además, se ha realizado una introducción al concepto de *streaming* y los diferentes usos y formatos que son utilizados para su funcionamiento.

DDS ofrece una gran potencia en cuanto a diseño gracias a la capacidad que ofrecen sus numerosos parámetros de calidad de servicio. Además permite un elevado nivel de concurrencia por lo que se plantea como una buena solución para implementar aplicaciones distribuidas y desacopladas de tiempo real, como es la transmisión de vídeo en directo.

Se ha probado la utilización del sistema, desarrollando una aplicación que permite el envío de vídeo y mensajes en directo. Esta aplicación gracias a DDS consigue que diferentes equipos remotos puedan visualizar un vídeo y mantener una comunicación textual mediante el envío de mensajes.

Además, existen muchas variantes a la hora de utilizar la aplicación, debido a la diversidad de posibilidades que ofrece la configuración inicial sobre ella. Se puede estar subscrito a un vídeo en modo Live a la vez que se está chateando en modo Tweet sobre tópicos totalmente diferentes, o recibir todo el vídeo completo mediante el modo VoD y únicamente recibir los últimos mensajes que se escribieron en el tópico del chat que hablaban de él mediante el modo Live.

En general, el desarrollo del proyecto ha sido el esperado encontrando ligeros problemas a la hora de obtener información en la forma de reproducir los datos de un *stream* con la herramienta FFmpeg, ya que, aún siendo muy potente, no está suficientemente documentada para sacarle el partido que se le puede sacar.

También surgieron algunos problemas con la transmisión de datos entre los publicadores y los subscriptores, a la hora de intentar conseguir una independencia entre los datos de chat y de vídeo, debido a la gran cantidad de configuraciones posibles dentro de un sistema DDS, además de problemas de memoria que surgieron cuando se intentaba obtener el vídeo en modo VoD, como se explica en el *Apéndice A.2*.

En definitiva, se han dado unos primeros pasos en el conocimiento y aprendizaje de una tecnología que puede permitir la consecución de aplicaciones que a día de hoy son complejas y difíciles de realizar y que mediante DDS y FFmpeg puede verse facilitado su desarrollo.

Además, no cabe duda que la transmisión de contenido multimedia será cada vez más importante. La tecnología de *streaming* es un mercado con futuro y grandes compañías ya están luchando por un trozo del pastel. La velocidad de Internet aumentará con el tiempo y con ella aumentará la calidad de las transmisiones, lo que conllevará a una mejor recepción de los contenidos multimedia.

6.2 Lineas futuras

A continuación se proponen una seria de mejoras que podrían dar un valor añadido a la misma:

- Mejora en reproducción de formatos. Tras verificar en las pruebas que con el formato avi no se obtienen buenos resultados, se debería intentar mejorar la visualización en este formato, así como incorporar algunos nuevos.

- Interfaz de vídeo mejorada. Dentro de la interfaz en la que se está reproduciendo el vídeo, una notable mejora, sería la inclusión de los clásicos componentes de reproducción como son el avance y retroceso del vídeo, en el caso en el que se esté visualizando un VoD, así como un regulador del volumen al que se está emitiendo el vídeo.

- Interfaz de mensajes de texto. La incorporación de interfaces diferenciadas dentro de la parte Chat de la aplicación, daría la posibilidad de diferenciar más claramente los mensajes que están siendo enviados y los que están siendo recibidos entre usuarios.

- Tópico Gestor. Esta es la mejora más ambiciosa que se propone. Consistiría en mantener un control de los usuarios y tópicos de chat y vídeo que se encuentran disponibles dentro del dominio. Para ello, como se muestra en la *Ilustración 44*, cada aplicación debería publicar y estar subscrito a un tópico común "Gestor".

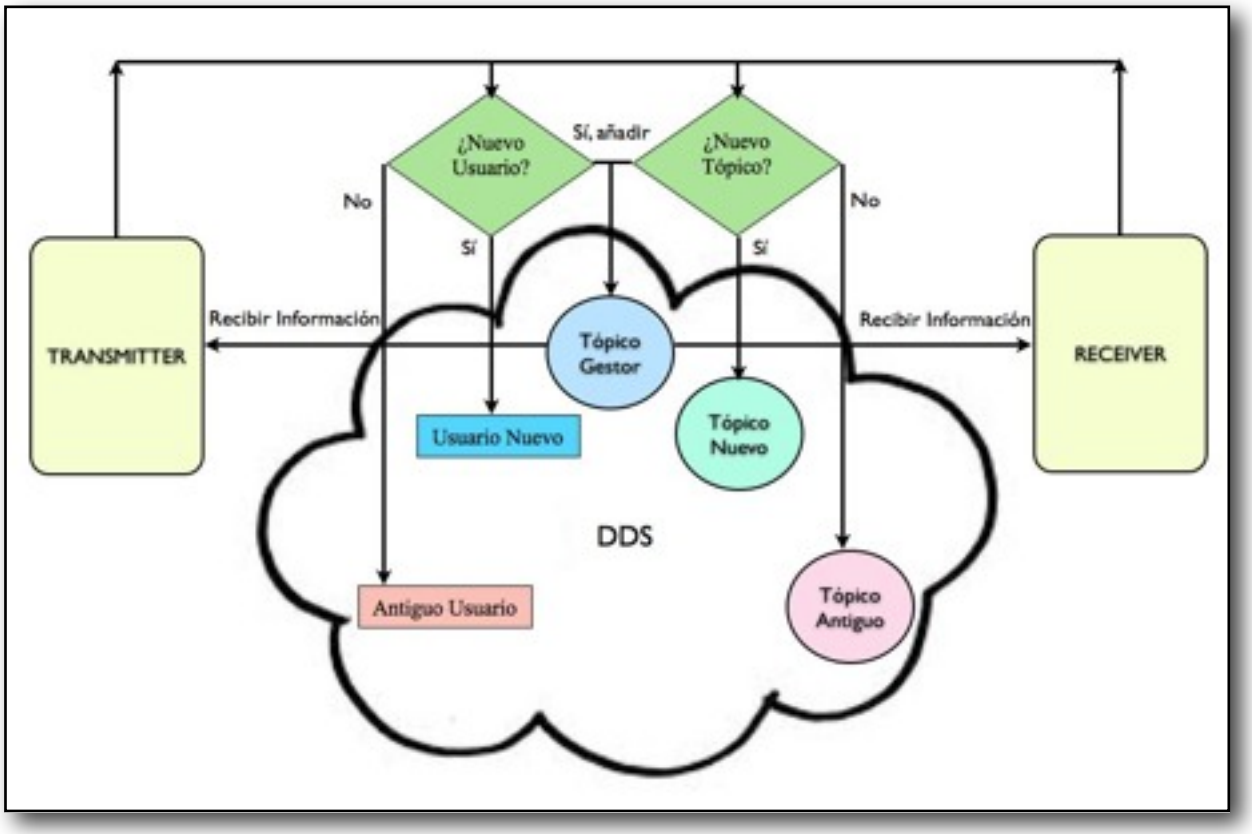


Ilustración 44. Diagrama del Tópico Gestor

Este tópico debería facilitar la información sobre los tópicos existentes, para saber a cuales te puedes subscribir o en cuales se pueden publicar, así como los nombres de los usuarios que ya están dados de alta en el dominio. Para ello se debe conocer el nombre con el que están dado de alta esos tópicos y usuarios, la configuración de QoS que poseen y una descripción sobre los datos que se publican o escriben.

Si no existiese el tópico que se quiere publicar, se debería primero, dar de "alta" en el tópico común "Gestor" con la información del mismo, y posteriormente en el dominio.

Capítulo 7:
Presupuesto del Proyecto

"Un presupuesto nos dice lo que no podemos pagar; pero ello no nos impide comprarlo".

William Feather *(1889-1981) Reportero del Cleveland Press*

En este capítulo se muestran los costes que ha tenido la realización del proyecto, así como, un diagrama con las diferentes actividades realizadas y el tiempo estimado que se ha tardado en realizarlas.

En la presente sección se presenta el presupuesto estimado para la realización de este proyecto, en el que se incluyen los costes tanto materiales como personales. Además se incluye el diagrama *Gantt* con el desglose de tareas, así como los recursos utilizados para el desarrollo del proyecto.

Para la realización del proyecto es necesario disponer de un ingeniero que realizar las tareas de documentación, análisis, diseño y desarrollo de la aplicación, así como las pruebas finales que verifiquen el proyecto.

Nombre	Fecha de inicio	Fecha de fin
T1.Estudio de las Tecnologías	1/09/10	20/09/10
T1.1.Estudio DDS-OpenSplice	1/09/10	20/09/10
T1.2.Estudio FFmpeg	1/09/10	20/09/10
T1.3.Estudio SDL	1/09/10	13/09/10
T2.Desarrollo de aplicaciones de iniciación	20/09/10	19/10/10
T2.1.Instalación del entorno	20/09/10	26/09/10
T2.2.Aplicación Prueba Chat	26/09/10	19/10/10
T2.3.Aplicación Prueba Video	26/09/10	19/10/10
T3.Desarrollo de la aplicación del PFC	19/10/10	2/07/11
T3.1.Instalación del entorno	19/10/10	27/10/10
T3.2.Desarrollo sistema DDS	27/10/10	29/01/11
T3.2.1.Desarrollo Publicador Chat-Video	27/10/10	29/12/10
T3.2.2.Desarrollo Subscriptor Chat-Video	27/11/10	27/01/11
T3.3.Desarrollo Video	29/01/11	2/07/11
T3.3.1.Desarrollo Recepción del Video	29/01/11	17/03/11
T3.3.2.Desarrollo Almacenamiento Video	1/03/11	26/05/11
T3.3.3.Desarrollo Reproducción Video	30/04/11	30/06/11
T4.Pruebas de Calidad	2/07/11	3/09/11
T5.Redacción de la Memoria	3/09/11	29/09/11

Ilustración 45. Listado de Tareas

Como se puede observar en la *Ilustración 45*, el proyecto ha sido dividido en cinco bloques realizados en todo momento por el ingeniero Juan Carlos Moreno. En las columnas de la derecha se muestran los tiempos de realización de cada una de las tareas.

A continuación, en la *Ilustración 46* se muestra el diagrama Gantt del proyecto realizado y las dependencias y relaciones que tienen unas tareas con otras.

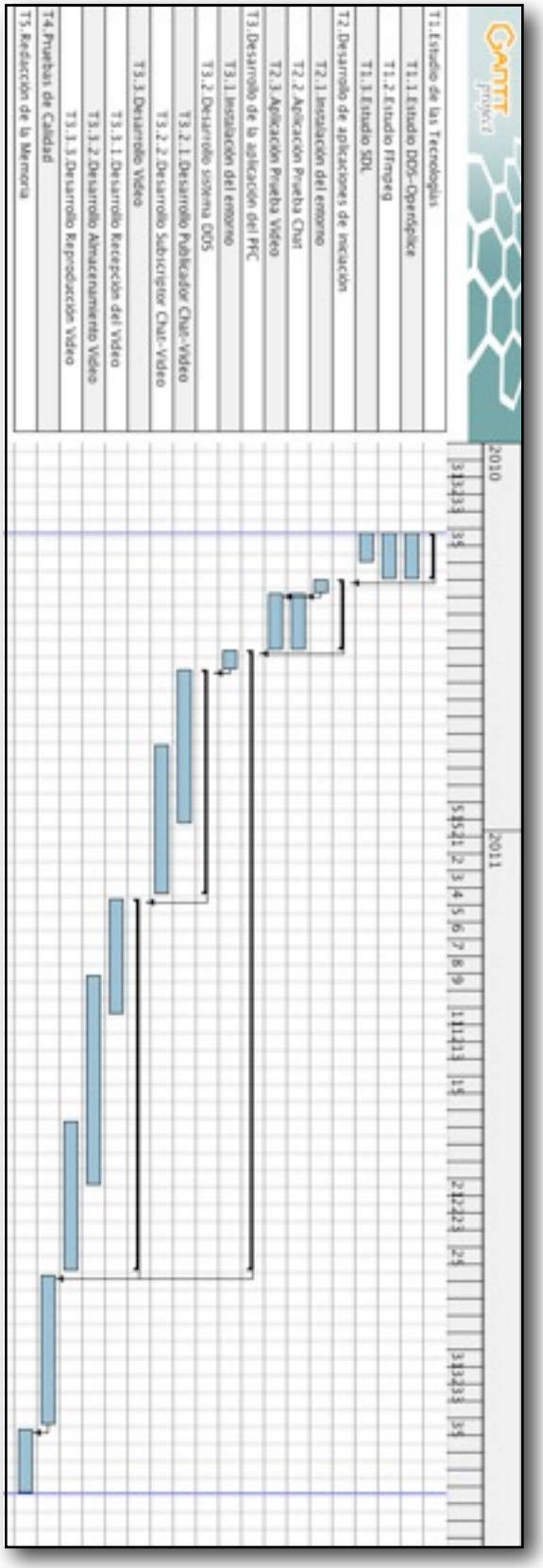


Ilustración 46. Diagrama Gantt del proyecto

A la hora de evaluar los costes del proyecto se diferencian 3 tipos que se detallan a continuación:

- **Personales**

Suponiendo un sueldo del ingeniero técnico de 20€/hora, y contando con que el desarrollador trabaja media jornada, y que el sueldo de un ingeniero superior son 30€/hora, el coste total del tiempo que se invierte en el proyecto es:

20€ x 4h = 80€/día. 80€ x 350 días (Eliminando los días de vacaciones) = 28000€
30€ x 2h = 60€/día. 60€ x 40 días = 2400

En total: 30400€.

- **Hardware**

Ordenador MacBook Pro para desarrollo = 1200€

Ordenador de sobremesa para pruebas = 800€

En total: 2000€.

- **Software**

En este caso todas las herramientas usadas para la realización del proyecto son gratuitas, tanto Sistema Operativo, como *middleware*, como las herramientas de *streaming*, etc.

En total: 0€.

Por lo que el presupuesto final para la realización del proyecto es de:

TOTAL = 32400€

BIBLIOGRAFÍA

[1] OMG Object Management Group. http://www.omg.org/ .Consultada en Septiembre 2010

[2] Toni A. Bishop, Ramesh K. Karne. "*A SURVEY of MIDDLEWARE*". 18th International Conference on Computers and Their Applications, March 2003, Honolulu, Hawaii, U.S.A

[3] Update on the origin of term "*middleware*". http://ironick.typepad.com/ironick/2005/07/update_on_the_o.html . Consultada en Enero de 2011

[4] David E. Bakken. "*Middleware*". Encyclopedia of Distributed Computing, Kluwer Academic Press, 2003.

[5] Peng, C, Chen, S, Chung, J, Roy-Chowdhury, A, and Srinivasan, V. "*Accessing existing business data from the World Wide Web*". IBM Systems Journal, Vol. 37, No. 1, 1998.

[6] José Luis Poza Luján. "*Revisión de los Sistemas de Comunicaciones más empleados en Control Distribuido*". Universidad Politécnica de Valencia. 10 Noviembre 2009.

[7] Coulouris, G., Dollimore, J., Kindberg, T. "*Sistemas Distribuidos, Conceptos y diseño*". Tercera Edición. Addison Wesley. Madrid. 2001.

[8] PrismTech Performance Critical Middleware. "*OpenSplice DDS, C Reference Guide*". Part Number: OS-CREFG Versión 4.1. Doc Issue 21, 15 April 2009.

[9] Raúl Castro Fernández. Proyecto Fin de Carrera: "*Desarrollo de software de videovigilancia para sistemas embarcados distribuidos con ICE"*. Universidad Carlos III de Madrid. 2009.

[10] Diccionario de la Lengua Española. Vigésimo Segunda Edición. "*http://www.rae.es/*". Consultada en Marzo 2011.

[11] José San Pedro Wandelmer. Tesis Doctoral: "*Arquitectura Paralela para el Procesamiento y Análisis de Vídeo Digital Utilizando Anotación MPEG-21. Aplicaciones Implantadas*". Universidad Politécnica de Madrid. 1 de Enero de 2006.

[12] Pietro Manzoni, Julio Pons, José Oliver. Documento de Asignatura de Transmisión de Datos Multimedia. "*Transmisión en Internet: Streaming de audio y vídeo*". Universidad Politécnica de Valencia. 2009

[13] Mauricio Venegas M., Aquiles Yáñez C., Agustín J. González. Publicación del Departamento de Electrónica. "*Transmisión de vídeo de alta calidad a través de redes IP utilizando herramientas de código abierto*". Universidad Técnica Federico Santa María. 2006.

[14] Nuria Sevilla Aguado. Proyecto Fin de Carrera: "*Diseño, implementación y lanzamiento de Codificador H.264*". Universidad San Pablo de CEU. Septiembre 2008

[15] Survive From Codecs Hell. http://vodinhphong.wordpress.com/2009/08/25/survive-from-codecs-hell/ .Consultada en Enero 2011

[16] MPEG-ORG. www.mpeg.org. Consultada en Febrero 2011

[17] VBrick Systems. "*MPEG-2 Transport vs. Program Stream*". Version R09242009. 2007

[18] Transport Stream: Enciclopedia Científica. Ciencia y Tecnología. http://e-ciencia.com/recursos/enciclopedia/Transport_Stream . Consultada en Febrero 2011

[19] PrimsTech Performance Critical Middleware. "*OpenSplice DDS C Tutorial Guide*". Part Number: OS-CTG Version 4.1. Doc Issue 19, 15 April 2009.

[20] FFmpeg. http://www.ffmpeg.org . Consultada en Septiembre 2010

[21] Theofilos Karachristos. Tesis Doctoral: "*A Real-Time Streaming Games on Demand System*". Athens Information Technology in collaboration with Carnegie Mellon University Information Networking Institute (INI). Abril 2007

[22] SDL Simple DirectMedia Layer. http://www.libsdl.org/. Consultada en Septiembre 2010

[23] Para qué se utilizan las aplicaciones *streaming*. http://culturacion.com/2010/12/para-que-se-utilizan-las-aplicaciones-streaming/ . Consultada en Mayo de 2011

[24] Joan Manuel Marquès i Puig, Xavier Vilajosana i Guillén, Pedro A. García López. "*Arquitecturas, paradigmas y aplicaciones de los sistemas distribuidos*". Universidad Oberta de Catalunya. FUOC • P07/M2106/02839

[25] Vídeo utilizado para las pruebas. "*White flag*" Primer sencillo del segundo álbum de Dido "*Life for rent*".

Discografía: Bertelsmann Music Group.

Autor: Dido Armstrong, Rollo Armstrong, Rick Nowels.

Productor: Dido Armstrong, Rollo Armstrong.

[26] Vídeo utilizado para las pruebas. "*End of Love*" álbum de Anna Abreu.

APÉNDICES

A. Instalaciones

A continuación se muestra como se realizan las instalaciones necesarias para el correcto funcionamiento del entorno de trabajo:

A.1 Instalación DDS OpenSplice y aplicación

La instalación de *OpenSplice* de *DDS* se hace a partir de los ficheros binarios disponibles en la página web de *PrismTech*. Realizaremos la instalación en el directorio */opt*. Para ello, una vez situados en dicho directorio, descargamos utilizando el comando *wget*.

```
$    cd /opt
```

```
$ wget ftp://www.opensplice.org/downloads/releases/OpenSpliceDDSV4.2-x86.linux2.6-gcc412-gnuc25-HDE.tar.gz
```

Una vez hecho esto, descomprimimos el fichero descargado.

```
$ tar xvzf OpenSpliceDDSV4.2-x86.linux2.6-gcc412-gnuc25-HDE.tar.gz
```

Una vez descargados y descomprimidos los ficheros binarios en el directorio */opt* debemos incluir los ficheros necesarios en las variables de entorno correspondientes. El fichero *release.com*, que se puede encontrar en el directorio */opt/HDE/x86.linux2.6*, nos ayudara a realizar dicho manejo.

Tal y como podemos ver a continuación, se trata de una serie de comandos que debemos ejecutar al inicio para que nuestros programas puedan encontrar en el PATH todos los ficheros necesarios:

```
echo "<<< OpenSplice HDE Release V4.2 For x86.linux2.6,
Date 2009-09-17 >>>"
OSPL_HOME="@@INSTALLDIR@@/HDE/x86.linux2.6"
OSPL_TARGET=x86.linux2.6
PATH=$OSPL_HOME/bin:$PATH
LD_LIBRARY_PATH=$OSPL_HOME/lib:$LD_LIBRARY_PATH
CPATH=$OSPL_HOME/include:$OSPL_HOME/include/sys:$CPATH
OSPL_TMPL_PATH=$OSPL_HOME/etc/idlpp
OSPL_URI=file://$OSPL_HOME/etc/config/ospl.xml
CLASSPATH=$CLASSPATH:$OSPL_HOME/jar/dcpssaj.jar
CLASSPATH=$CLASSPATH:$OSPL_HOME/jar/dcpscj.jar
export OSPL_HOME OSPL_TARGET PATH LD_LIBRARY_PATH CPATH
OSPL_TMPL_PATH OSPL_URI CLASSPATH
```

Con este objetivo, añadiremos las siguientes líneas del *script* al final del fichero de configuración */etc/profile*:

```
OSPL_HOME="/opt/HDE/x86.linux2.6"
OSPL_TARGET=x86.linux2.6
PATH=$OSPL_HOME/bin:$PATH
LD_LIBRARY_PATH=$OSPL_HOME/lib:$LD_LIBRARY_PATH
CPATH=$OSPL_HOME/include:$OSPL_HOME/include/sys:$CPATH
```

OSPL_TMPL_PATH=$OSPL_HOME/etc/idlpp

OSPL_URI=file://$OSPL_HOME/etc/config/ospl.xml

CLASSPATH=$CLASSPATH:$OSPL_HOME/jar/dcpssaj.jar

CLASSPATH=$CLASSPATH:$OSPL_HOME/jar/dcpscj.jar

export OSPL_HOME OSPL_TARGET PATH LD_LIBRARY_PATH CPATH OSPL_TMPL_PATH OSPL_URI CLASSPATH

Si queremos que la terminal actual cargue esta nueva configuración debemos introducir el siguiente comando:

$ source /etc/profile

El ultimo paso para configurar nuestro entorno será incluir las bibliotecas dinámicas de OpenSplice DDS en el directorio */etc/ld.so.conf.* Si nuestra distribución tiene organizadas las bibliotecas que se incluyen dentro de este fichero en ficheros distintos en la carpeta */etc/ld.so.conf.d/* incluiremos un nuevo fichero llamado *opensplicedds.conf* cuyo contenido será:

/opt/HDE/x86.linux2.6/lib

Para actualizar nuestro entorno basta con ejecutar el siguiente comando:

$ ldconfig

Una vez instalado la configuración por defecto que trae el sistema viene definido en el fichero *ospl.xml:*

```
<OpenSpliceDDS>
    <Domain>
        <Name>OpenSpliceDDSV3.3</Name>
        <Database>
            <Size>10485670</Size>
        </Database>
        <Lease>
            <ExpiryTime update_factor="0.5">60.0</ExpiryTime>
        </Lease>
        <Service name="networking">
            <Command>networking</Command>
        </Service>
        <Service name="durability">
            <Command>durability</Command>
        </Service>
    </Domain>

    <NetworkService name="networking">
        <Partitioning>
            <GlobalPartition Address="broadcast"/>
        </Partitioning>
        <Channels>
            <Channel name="BestEffort" reliable="false" default="true">
                <PortNr>53340</PortNr>
            </Channel>
            <Channel name="Reliable" reliable="true">
                <PortNr>53350</PortNr>
            </Channel>
        </Channels>
        <Discovery enabled="true">
            <PortNr>53360</PortNr>
        </Discovery>
    </NetworkService>

    <DurabilityService name="durability">
        <Network>
            <InitialDiscoveryPeriod>2.0</InitialDiscoveryPeriod>
```

```
            <Alignment>
                <TimeAlignment>FALSE</TimeAlignment>
                <RequestCombinePeriod>
                    <Initial>2.5</Initial>
                    <Operational>0.1</Operational>
                </RequestCombinePeriod>
            </Alignment>
            <WaitForAttachment maxWaitCount="10">
                <ServiceName>networking</ServiceName>
            </WaitForAttachment>
        </Network>
        <NameSpaces>
            <NameSpace durabilityKind="Durable"
                    alignmentKind="Initial_and_Aligner">
                <Partition>*</Partition>
            </NameSpace>
        </NameSpaces>
    </DurabilityService>
</OpenSplice>
```

Para que la máquina pueda ejecutar cualquier programa publicador o suscriptor con OpenSplice DDS, debe arrancarse primero el programa *ospl* con el siguiente comando,

```
$ ospl start
```

Para parar el programa *ospl* se utilizará el siguiente comando:

```
$ ospl stop
```

De esta manera estaremos en condiciones de probar nuestra aplicación y de comenzar a desarrollar aplicaciones utilizando esta implementación libre de DDS.

Para crear los ejecutables de la aplicación, desde la carpeta raíz donde se encuentra el código del proyecto, se debe de ejecutar el archivo *Makefile* para que se compilen las funciones y así crear los ejecutables dentro de la carpeta "exec".

Dentro de dicha carpeta se encuentran los dos ejecutables, el Transmitter y el Receiver. Ambos deben ser lanzados en terminales diferentes mediante los siguientes comandos:

```
$ ./Transmitter 2>Transmitter.log

$ ./Receiver 2>Receiver.log
```

Estas sentencias permiten que las trazas del programa se guarden en el fichero ".log" creado y que se inicie cada una de las aplicaciones, en las cuales se tendrá que realizar la configuración inicial descrita en los apartados 4.4.2 y 4.4.3.

A.2 Modificaciones necesarias para el funcionamiento de la aplicación

Para el correcto funcionamiento de la aplicación, en el modo VoD, se deben realizar las siguientes modificaciones:

Por defecto, como se ha visto en el fichero *ospl.xml*, el sistema de OpenSplice tiene una memoria de 10 mb, por lo que cuando se mantienen un número menor de bytes en el sistema distribuido no existe ningún problema, pero en el caso que se superen, se produce un error de memoria.

Para el caso del tópico de chat, con los 10 mb es suficiente para mantener los datos de los mensajes enviados, pero en el caso del tópico de vídeo, no es suficiente en el modo VoD, ya que la memoria debería ser mayor que el vídeo que estamos transmitiendo para que se mantuviesen todos los datos del vídeo

en el sistema y usuarios conectados posteriormente al envío del vídeo pudiesen recibirlo completamente.

En nuestro caso el vídeo ocupa 54 mb, por lo que habría que aumentar la memoria a ese tamaño. Para realizar esto hay que modificar la memoria compartida que tiene la máquina Linux en la que estamos trabajando, esto se realiza, con privilegios de administrador, del siguiente modo:

Para comprobar el valor actual:

```
$ /sbin/sysctl -a | grep shmmax
kernel.shmmax = 33554432
```

Para modificar el tamaño de la memoria física de nuestro sistema se usa el siguiente comando:

```
$ /sbin/sysctl kernel.shmmax=6442450944
kernel.shmmax = 6442450944
```

Además se debe modificar la memoria del sistema OpenSplice en el fichero *ospl.xml* de configuración:

```
<Database>
    <Size> 6442450944 </Size>
</Database>
```

A.3 Instalación FFmpeg

Para instalar de forma exitosa la herramienta ffmpeg bajo el sistema operativo Ubuntu/Linux se debe realizar el siguiente procedimiento.

Descargar e instalar una serie de paquetes:

$ sudo apt-get install dpkg-dev libimlib2-dev texi2html liblame-dev libfaad2-dev libmp4v2-dev

$ sudo apt-get install libfaac-dev libxvidcore4-dev libtheora-dev libgsm1-dev libogg-dev libvorbis-dev

$ sudo apt-get install liba52-dev libdts-dev libsdl1.2-dev libraw1394-dev libdc1394-13-dev quilt

$ sudo apt-get install subversion build-dep

Una vez realizado esto, se realiza la descarga e instalación de la herramienta ffmpeg:

$ sudo apt-get ffmpeg

Luego se obtiene el código fuente a través de un cliente SVN, el comando para ello es el siguiente:

$ svn checkout svn://svn.ffmpeg.org/ffmpeg/trunk ffmpeg

Con esto se descarga la carpeta ffmpeg en la ubicación actual, dentro de ella se encuentra todo el código fuente de la herramienta ffmpeg.

Luego se ingresa a la carpeta descargada:

$ cd ffmpeg

Y dentro de ella, con el fin de evitar posibles problemas, se ejecuta el siguiente comando:

$ make distclean

Luego se especifican las opciones para la configuración del ffmpeg, estas opciones son diversas, entre ellas se cuenta con:

–enable-gpl –enable-pp –enable-x11grab –enable-libvorbis –enable-libtheora –enable-liba52 –enable-libdc1394 –enable-libgsm –enable-libmp3lame –enable-libfaad –enable-libfaac –enable-libxvid –enable-pthreads –enable-libx264 –enable-shared

Para el trabajo que se menciona en este documento, sólo es necesario ejecutar el siguiente comando de configuración:

$ ./configure –enable-gpl –enable-libmp3lame –enable-x11grab

Una vez realizada la configuración de compilación del ffmpeg, se ejecutan los siguientes comandos:

$ sudo make

$ sudo make install

A.4 Instalación SDL

Los paquetes más importantes a instalar son:

- libsdl1.2debian: paquete de librerías SDL
- libsdl1.2-dev: paquete de librerías SDL para desarrollo.
- libsdl-image1.2: paquete para trabajar con varios tipos de imágenes.
- libsdl-image1.2-dev: paquetes para desarrollo
- libsdl-mixer1.2: paquete para trabajar con sonidos.
- libsdl-mixer1.2-dev: paquetes para desarrollo

Para la instalación se deben introducir los siguientes comandos en la consola:

```
$ apt-get install libsdl1.2debian
$ apt-get install libsdl1.2-dev
$ apt-get install libsdl-image1.2
$ apt-get install libsdl-image1.2-dev
$ apt-get install libsdl-mixer1.2
$ apt-get install libsdl-mixer1.2-dev
```

A.5 Instalación en dispositivo Maemo

A continuación se detallan los pasos que se realizaron para la inclusión de la aplicación en un terminal Maemo N810 con fallidos resultados.

Lo primero que se realizó fue el intento de incluir OpenSplice en el emulador de Maemo que tiene instalada la máquina virtual Linux. Para ello se siguieron los comunes a la instalación en un entorno de PC, modificando las rutas por las de el emulador:

Se descomprime el fichero tar.gz con los datos binarios de OpenSplice dentro de la ruta:

/home/teleco/.maemo-sdk/rootstraps/armel/diablo4.1.2_armel/opt

Tras descomprimirlo, se configura el fichero *profile* del emulador que se encuentra en */*home/teleco/.maemo-sdk/rootstraps/armel/diablo4.1.2_armel/etc con los siguientes datos:

OSPL_HOME="/opt/HDE/x86.linux2.6"

OSPL_TARGET=x86.linux2.6

PATH=$OSPL_HOME/bin:$PATH

LD_LIBRARY_PATH=$OSPL_HOME/lib:$LD_LIBRARY_PATH

CPATH=$OSPL_HOME/include:$OSPL_HOME/include/sys:$CPATH

OSPL_TMPL_PATH=$OSPL_HOME/etc/idlpp

OSPL_URI=file://$OSPL_HOME/etc/config/ospl.xml

CLASSPATH=$CLASSPATH:$OSPL_HOME/jar/dcpssaj.jar

CLASSPATH=$CLASSPATH:$OSPL_HOME/jar/dcpscj.jar

export OSPL_HOME OSPL_TARGET PATH LD_LIBRARY_PATH CPATH OSPL_TMPL_PATH OSPL_URI CLASSPATH

El primer problema aparece cuando se quiere dar la orden para actualizar los datos, con el comando especial *sb2 -e* necesario para las modificaciones que se hagan en el terminal Maemo:

sb2 -e source /etc/profile

Obteniendo el siguiente error:

whoami: cannot find username for UID 1000
whoami: cannot find username for UID 1000
mkdir: cannot create directory `/home//.osso': Permission denied
cp: cannot create regular file `/home//.osso/current-gtk-theme': No such file or directory
cp: cannot create regular file `/home//.osso/current-gtk-key-theme': No such file or directory

Aún obteniendo el error, se continúa con la instalación avanzando en el siguiente paso que se debe realizar, introducir las librerías dentro del fichero *ld.so.conf* que se encuentra en /home/teleco/.maemo-sdk/rootstraps/armel/diablo4.1.2_armel/etc.

El segundo problema aparece al intentar actualizar los datos de este fichero mediante el comando:

sb2 -e ldconfig

Obteniendo el siguiente error:

/home/teleco/.maemo-sdk/rootstraps/armel/diablo4.1.2_armel/sbin/ldconfig:

Can't remove old temporary cache file /home/teleco/.maemo-sdk/rootstraps/armel/diablo4.1.2_armel/etc/ld.so.cache~: Read-only file system

Tras la supuesta instalación del programa intentamos iniciar la aplicación OpenSplice mediante el comando:

sb2 -eR ospl start

Obteniendo el consiguiente error, al no haber conseguido realizar las instalaciones previas correctamente:

ospl: error while loading shared libraries: libddsos.so: cannot open shared object file: No such file or directory

Tras el intento fallido de la aplicación OpenSplice en el emulador, se intenta realizar la instalación directamente en un terminal físico, pero al realizar las instalaciones anteriores dentro del terminal, e intentar iniciar la aplicación se obtiene un error de sintaxis.

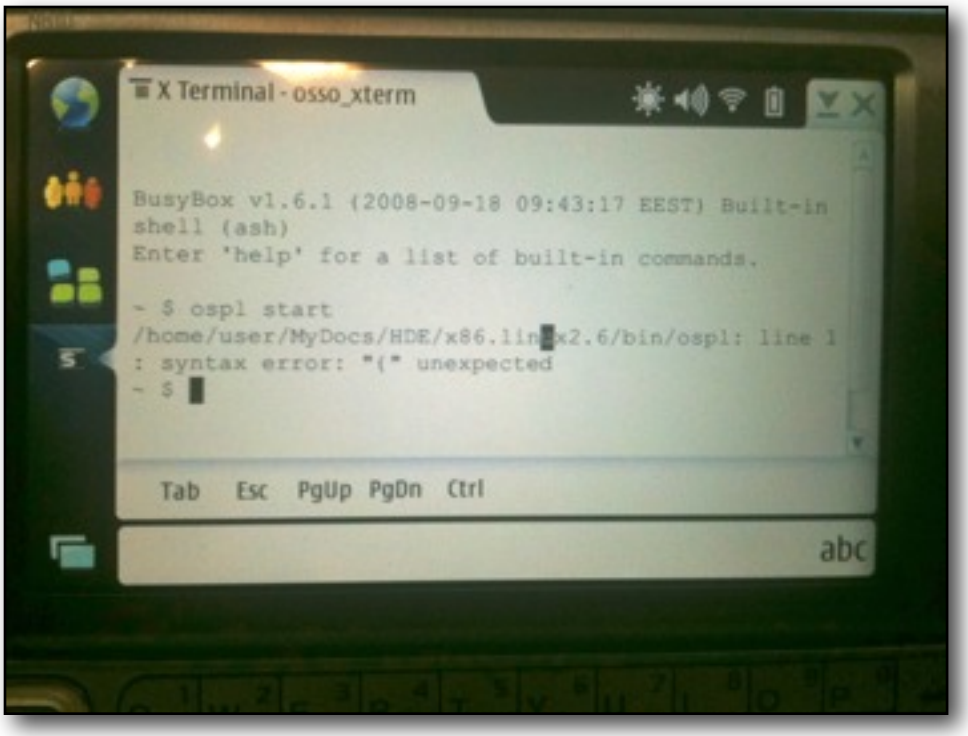


Ilustración 47. Instalación OpenSplice en Maemo

Como se observa en la *Ilustración 47*, el terminal no inicia OpenSplice, si no que devuelve un mensaje de error en el que parece no interpretar correctamente el binario que arranca la aplicación.

Tras los intentos fallidos de instalación se intentó buscar respuestas a esta problemática directamente preguntando en el foro que tiene abierto OpenSplice en la página web http://forums.opensplice.org/index.php?/index introduciendo un nuevo tema en el que intentar obtener respuesta sobre una posible instalación en un terminal Maemo:

http://forums.opensplice.org/index.php?/topic/1442-opensplice-for-maemo/page__p__2029__hl__Maemo__fromsearch__1&#entry2029

A día de hoy, no se ha obtenido respuesta, por lo que no se ha continuado por esta línea de trabajo debido a la imposibilidad de unir la tecnología de OpenSplice con Maemo.